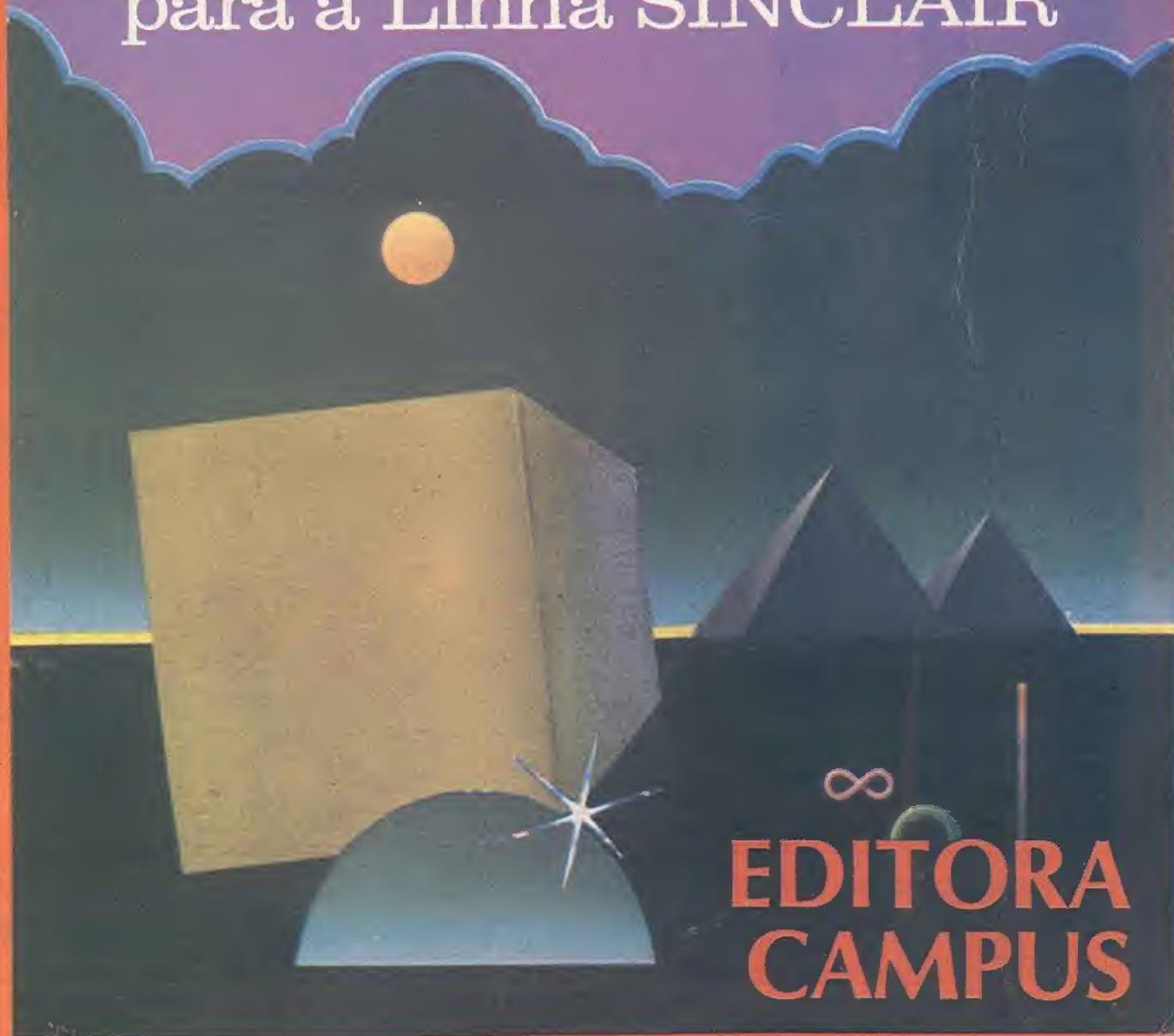


NELSON N. S. SANTOS

Além do BASIC

Linguagem ASSEMBLY
para a Linha SINCLAIR



 **EDITORAS**
CAMPUS

TÍTULOS DE INTERESSE CORRELATO

ADA®: UMA INTRODUÇÃO – *H. Ledgard*

BASIC BÁSICO, 5ª ed. – *J. C. Pereira Filho*

BASIC COM ESTILO – *P. Nagin e H. F. Ledgard*

BASIC PARA APLICAÇÕES COMERCIAIS – *D. Hergert*

BASIC PARA MICROS PESSOAIS, 2ª ed. – *J. C. Pereira Filho*

BASIC RÁPIDO: ALÉM DO BASIC TRS-80 – *G. A. Gratzer e T. G. Gratzer*

BASIC SINCLAIR – *R. U. Christmann*

COBOL COM ESTILO – *L. J. Chmura e H. F. Ledgard*

COBOL PARA ESTUDANTES, 4ª ed. – *A. Parkin*

COBOL: REGRAS PARA PROGRAMADORES – *G. Ledin Jr. et al.*

CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO – *C. Ghezzi e M. Jazayeri*

ENCICLOPÉDIA DA LINGUAGEM BASIC – *C. Pereira e R. B. Alcantara*

FORTRAN PARA MICROS – *G. Marshall*

GUIA DE LINGUAGENS DE COMPUTADORES – *H. L. Helms Jr.*

INTRODUÇÃO À PROGRAMAÇÃO FORTRAN – *J. C. Pereira Filho*

JCL SISTEMA/370, 4ª ed. – *G. D. Brown*

LINGUAGEM DE PROGRAMAÇÃO ALGOL, 2ª ed. – *L. M. Segre*

LINGUAGEM PASCAL, 2ª ed. – *V. L. Strube de Lima*

LOGO – *P. Goodyear*

MANUAL DE COBOL ESTRUTURADO – *D. D. McCracken*

MANUAL DE LINGUAGEM C – *L. Hancock e M. Krieger*

**PROGRAMAÇÃO EM ASSEMBLER E LINGUAGEM DE MÁQUINA, 2ª ed.
– *D. C. Alexander***

PROGRAMAÇÃO EM BASIC – *J. G. Kemeny e T. Kurtz*

RPG II – *J. C. Pereira Filho*

Conheça toda a linha de Informática da Editora Campus, com títulos nas áreas de:
Introdução à Computação; Teoria, Organização e Processamento de Dados;
Linguagens; Programação; Programas e Aplicativos; Sistemas Operacionais e
Compiladores; Arquitetura e Equipamentos; Interesse Especial.

MICROBITS

Macroinformações e Programas para Micros TK 82-83-85, CP-200 e compatíveis.
Uma publicação bimestral com análises de hardware, aplicativos, artigos sobre
linguagem de máquina, dicas de programação e respostas para suas dúvidas.

Procure nossas publicações nas boas livrarias ou comunique-se diretamente com:
EDITORAS CAMPUS LTDA.

Livros Científicos e Técnicos

Rua Barão de Itapagipe 55

20261 — Rio de Janeiro — RJ — Brasil

Telefone.: (021) 284 8443

Atendemos também pelo reembolso postal

**Além do
BASIC**

NELSON N. S. SANTOS
PROFESSOR/ENGENHEIRO

Além do BASIC

Linguagem ASSEMBLY
para a Linha SINCLAIR

EDITORIA CAMPUS LTDA.
Rio de Janeiro

© 1985, Editora Campus Ltda.

Todos os direitos reservados e protegidos pela Lei 5988 de 14/12/1973.
Nenhuma parte deste livro poderá ser reproduzida ou transmitida sejam quais forem os meios empregados: eletrônicos, mecânicos, fotográficos, gravação ou quaisquer outros.

Todo o esforço foi feito para fornecer a mais completa e adequada informação. Contudo a editora e o(s) autor(es) não assumem responsabilidade alguma pelos resultados e uso da informação fornecida.

Recomendamos aos leitores, em consequência, testar toda a informação antes de sua efetiva utilização.

A Editora Campus não é filiada a nenhum fabricante de sistemas computacionais.

Capa
Otavio Studart

Diagramação, composição, paginação e revisão
Editora Campus Ltda.
Rua Barão de Itapagipe 55 Rio Comprido
Tel.: (021) 284 8443
20261 Rio de Janeiro RJ Brasil
Endereço telegráfico: CAMPUSRIO

ISBN 85-7001-247-0

Ficha Catalográfica
CIP-Brasil. Catalogação-na-fonte
Sindicato Nacional dos Editores de Livros, RJ.

Santos, Nelson N. S., 1949-
S236a Além do BASIC : linguagem ASSEMBLY para a linha Sinclair
/ Nelson N. S. Santos. — Rio de Janeiro : Campus, 1985.

Apêndices
ISBN 85-7001-247-0.

1. Assembly Z80 (Línguagem de programação para computadores) 2. Assembly Z80 (Línguagem de programação para computadores) — Problemas, exercícios etc. 3. TK (Computador) — Programação I. Título

85-0198

CDD. — 001.642
001.6424

**“Eu falo espanhol com Deus,
italiano com as mulheres,
francês com os homens e
alemão com o meu cavalo.”**

*Carlos V, Imperador do
Sacro Império Romano-Germânico
(1500-1558)*

Se você já fala BASIC com seu micro SINCLAIR, fale ASSEMBLY direto com o
microprocessador Z-80 — ele entende . . .

Agradecimentos a

- meus pais;
- minha irmã;
- minha sogra;
- minha tia Maria Adelaide;
- D. Irany, que cuida de nossa casa e de todos nós.

Cada um de vocês, de certa maneira, tornou este livro possível.
Obrigado.

Para

NOEMIA, minha mulher, co-autora pelo estímulo;
DIANA,
HOMERO,
CYNTHIA MARIA,
CAROLINA e
ARTHUR DANIEL, meus filhos;

este livro é dedicado.

CAPÍTULO 1 IMPRIMINDO NA TELA

- 1.1 Introdução, 14
- 1.2 Substituindo o LET, 14
- 1.3 Exercícios, 15
- 1.4 Substituindo o PRINT, 16
- 1.5 Voltando ao BASIC, 17
- 1.6 O primeiro programa ASSEMBLY, 18
- 1.7 O que fazer com ele?, 18
- 1.8 O carregador ASSEMBLY, 21
- 1.9 De volta ao primeiro programa ASSEMBLY, 22
- 1.10 Alterando o caractere, 23
- 1.11 Exercícios, 24
- 1.12 Chamando rotinas na ROM, 24
- 1.13 Exercícios, 25

CAPÍTULO 2 ENCHENDO A TELA

- 2.1 Introdução, 28
- 2.2 Salto absoluto incondicional, 28
- 2.3 Enchendo a tela, 29
- 2.4 Exercícios, 29
- 2.5 Salto relativo incondicional, 30
- 2.6 Exercícios, 31

CAPÍTULO 3 ESCREVENDO STRINGS NA TELA

- 3.1 Introdução, 34
- 3.2 Exercícios, 34
- 3.3 Pares de registradores, 35
- 3.4 Exercícios, 36
- 3.5 Colocando uma string na tela, 36
- 3.6 Exercícios, 38

CAPÍTULO 4 ESCOLHENDO O LOCAL DE IMPRESSÃO

- 4.1 A sub-rotina PRINT AT, 40
- 4.2 Exercícios, 41
- 4.3 Combinando sub-rotinas da ROM, 41
- 4.4 Exercícios, 41

CAPÍTULO 5 FAZENDO LOOPS – 1^a PARTE

- 5.1 Introdução, 44
- 5.2 A função USR, 44
- 5.3 Mais uma instrução ASSEMBLY-NOP, 46
- 5.4 Exercícios, 47
- 5.5 Fazendo LOAD de um registrador para outro, 48
- 5.6 Exercícios, 49
- 5.7 Aprendendo a somar, 49
- 5.8 Exercícios, 50
- 5.9 LOOPS – O conceito de contador, 51
- 5.10 Exercícios, 53

CAPÍTULO 6 FAZENDO LOOPS – 2^a PARTE

- 6.1 Introdução, 61
- 6.2 Instruções INC r e DEC r, 61
- 6.3 Saltos absolutos condicionais, 63
- 6.4 Saltos relativos condicionais, 63
- 6.5 A instrução COMPARE (CP), 64
- 6.6 Uso de contadores, 64
- 6.7 Exercícios, 65

CAPÍTULO 7 A TELA DE TV

- 7.1 Introdução, 71
- 7.2 Alterando as linhas de edição, 72
- 7.3 A memória de tela, 74
- 7.4 Exercícios, 76
- 7.5 Novas instruções ASSEMBLY, 77
- 7.6 Usando o arquivo de tela, 78
- 7.7 Exercícios, 81

CAPÍTULO 8 MISCELÂNEA DE EXERCÍCIOS

- 8.1 Introdução, 86
- 8.2 Exercícios, 86
- 8.3 Instruções que usam a pilha, 96
- 8.4 O último exercício: O espelho, 97

CAPÍTULO 9 CLS ESPIRAL

- 9.1 Introdução, 109
- 9.2 Fazendo pausa em linguagem de máquina, 109
- 9.3 Exercícios, 110
- 9.4 Mais uma instrução, 112
- 9.5 Exercício, 113
- 9.6 O CLS espiral, 113
- 9.7 Exercícios, 114
- 9.8 Palavras finais, 114

- APÊNDICE 1 O MICROPROCESSADOR Z-80, 120
- APÊNDICE 2 O CONJUNTO DE CARACTERES, 124
- APÊNDICE 3 TABELA DE CONVERSÃO DECIMAL/HEXADECIMAL, 129
- APÊNDICE 4 TABELA DE COMPLEMENTO DE DOIS, 130
- APÊNDICE 5 AS VARIÁVEIS DO SISTEMA, 131
- APÊNDICE 6 EQUIVALÊNCIA DE NOMENCLATURA DAS VARIÁVEIS DO SISTEMA, 133
- APÊNDICE 7 USANDO AS VARIÁVEIS DO SISTEMA, 134
- APÊNDICE 8 PALAVRAS RESERVADAS, 137
- APÊNDICE 9 O CONJUNTO DE INSTRUÇÕES DO Z-80 – LISTA NUMÉRICA, 138
- APÊNDICE 10 O CONJUNTO DE INSTRUÇÕES DO Z-80 – LISTA ALFABÉTICA, 146
- APÊNDICE 11 FLAGS AFETADAS PELAS INSTRUÇÕES, 154
- APÊNDICE 12 PROGRAMA BASIC PARA VER CÓDIGOS DE MÁQUINA, 157
- APÊNDICE 13 CARREGADOR ASSEMBLY, 158
- APÊNDICE 14 O MAPA DA MINA (UM PEQUENO ROTEIRO DA ROM), 159
- APÊNDICE 15 TESTE DA ROM DO SEU SINCLAIR, 162
- APÊNDICE 16 MISCELÂNEA, 163

- *Gatinho de Cheshire — começou ela, bastante timidamente ... — Você pode me dizer, por favor, qual o caminho que devo tomar?*
- *Depende bastante de para onde você quer ir — disse o Gato.*
- *Não me importa muito — disse Alice.*
- *Então, não importa qual o caminho que você tome — disse o Gato.*

Lewis Carroll,

em *Alice's Adventures in Wonderland* (1865)

Mais um livro sobre linguagem de máquina para a linha Sinclair?

Sim. Mas nossa situação ao nos propormos a escrever este livro era radicalmente diferente da de Alice — nós sabíamos onde queríamos chegar:

Foi nosso propósito colocar toda a experiência adquirida ao longo de quinze anos de magistério a serviço de criar um livro que atendesse a alguns pontos básicos. Desejávamos um livro que:

- fosse didático e compreensível;
- fosse agradável de ler;
- tivesse muitos exercícios;
- os exercícios tivessem respostas;
- os programas não apresentassem erros de impressão, e rodassem realmente;
- tivesse muitas rotinas em linguagem de máquina realmente úteis, e explicasse os comos e os porquês delas funcionarem;
- aclarassee os mistérios, em vez de aumentá-los ...

Uma vez fixados os objetivos, nos esforçamos bastante para atingi-los.

Conseguimos? Cabe ao leitor a palavra final.

O AUTOR.

Introdução

1.1

INTRODUÇÃO

Para imprimir o caractere * no canto superior esquerdo da tela, o seguinte programa BASIC poderia ser usado.

```
10 LET A = 23  
20 PRINT CHR$ A;
```

A linha 10 do programa define uma variável A, e a ela atribui o valor 23.

A linha 20 imprime na tela o caractere cujo código foi definido pela variável A (23 é o código decimal do caractere asterisco — confira no Apêndice 2).

Nosso primeiro programa ASSEMBLY fará exatamente a mesma coisa (imprimir um asterisco no canto superior esquerdo da tela), e terá exatamente a mesma estrutura.

1.2

SUBSTITUINDO O LET

Em BASIC Sinclair, para atribuir valores a variáveis, usamos o comando LET. Assim,

```
10 LET A = 23
```

define a variável A e a ela atribui o valor 23.

Num programa BASIC, você pode ter um número muito grande de variáveis, e a elas atribuir o nome que desejar (a variável poderia ser definida por 10 LET BRUXA = 23 ...)

Em ASSEMBLY não é bem assim.

O microprocessador Z-80 possui vários registradores internos, dos quais 7 serão estudados inicialmente (o Apêndice 1 faz um estudo detalhado do microprocessador Z-80).

São eles A, B, C, D, E, H e L.

O primeiro (A) é chamado de acumulador e é, sob vários aspectos, um registrador privilegiado. Breve você verá o porquê.

Cada um destes registradores pode ser carregado com um byte, ou seja, com um número ("palavra") de 8 bits.

Usando hexadecimal, vemos que cada registrador pode ser carregado com um número de 00 até FF (ou seja, de 0 a 255 em decimal).

Vamos agora aprender a carregar (LOAD) estes registradores. (Claro que este LOAD *nada tem a ver* com o comando BASIC LOAD.)

A instrução tem a seguinte forma geral:

LD r,N

onde

LD → mnemônico de LOAD

r → registrador de 8 bits (A, B, C, D, E, H, L)

N → número entre 0 e 255 d

Aqui um detalhe importante: quando falarmos sobre uma instrução, usaremos números decimais (são os números nos quais nosso cérebro está acostumado a pensar). Estes números decimais, para serem introduzidos no programa, devem ser convertidos em hexadecimal. Use e abuse da tabela do Apêndice 3.

Assim, a correspondência é simples:

ASSEMBLY	BASIC
----------	-------

LD A,23	LET A = 23
---------	------------

LD A,32	LET A = 32
---------	------------

A instrução "custa" 2 bytes: um para a instrução propriamente dita e outro para o número.

A tabela abaixo fornece os códigos hexadecimais das instruções.

LD A,N	3E ..
LD B,N	06 ..
LD C,N	0E ..
LD D,N	16 ..
LD E,N	1E ..
LD H,N	26 ..
LD L,N	2E ..

Obs.: O conjunto de instruções do Z-80 está nos Apêndices 9 e 10.

1.3

EXERCÍCIOS

- 01 Primeiramente vamos ver se você entendeu como usar a tabela do Apêndice 3.
Complete as lacunas abaixo.

	HEXADECIMAL	DECIMAL
a)	F0
b)	0F
c)	0
d)	121
e)	39
f)	FF

- 02 Suponha que seja necessário carregar o acumulador A com o código do caractere*. A instrução é

LD A,23

e seu código hexadecimal é

3E 17

pois {3E equivale a LD A,N (tabela na página anterior)
17h = 23d (tabela do Apêndice 3)}

Complete então as lacunas abaixo:

	INSTRUÇÃO	CÓDIGO HEXA
a)	LD A,151
b),.....	06 20
c)	LD L,..... 10
d)	LD C,240
e)	26 FF
f)	LD E,..... 80
g),8	16.....

- 03 Seria possível a instrução LD B,704? Por quê?

- 04 Mostre que você está firme. Sem consultar o texto (nem tabela alguma), complete as lacunas:

- Os 7 registradores que já aprendemos a carregar são, em ordem alfabética, , , , , e
- O maior valor que pode ser carregado em um registrador é (decimal) ou (hexa).
- LD é o mnemônico para, que significa, ou seja, armazenar informação.
- O registrador A leva o nome especial de
- Escreva a instrução ASSEMBLY (não os códigos hexa) para carregar o registrador A com o valor hexa FF:.....,.....

1.4

SUBSTITUINDO O PRINT

O programa BASIC que vimos no item 1.1 tinha na linha 10 a instrução LET, e já vimos como substituí-la em ASSEMBLY.

Na linha 20, a instrução PRINT. Vamos aprender agora como substituí-la.

Existem instruções chamadas RESTART (mnemônico: RST) que estão colocadas em lugar fixo na ROM, nos endereços (hexa) 0, 8, 10, 18, 20, 28, 30 e 38. Vamos enfatizar: estes endereços *estão em hexadecimal*.

A função das instruções RESTART é variável de micro para micro que utiliza o microprocessador Z-80. Por exemplo, na família TRS-80 (CP-500, DGT-100 etc.) as

instruções RST têm os mesmos códigos hexa, estão nos mesmos endereços da ROM, mas ... fazem outras coisas!

Nos micros de lógica SINCLAIR a função das instruções RESTART é a seguinte:

RST 0 — inicializa o sistema, é ativada quando se liga o computador. Veja os Apêndices 14 e 16 para mais detalhes.

RST 8 — fornece o código do erro cometido (rotina de erro).

RST 10 — coloca no vídeo o caractere cujo código está no acumulador.

RST 18 — obtém um caractere da linha de BASIC.

RST 20 — obtém o próximo caractere da linha de BASIC.

RST 28 — salta para a rotina de cálculo com ponto flutuante.

RST 30 — organiza o espaço na memória.

RST 38 — rotina de interrupção para mostrar uma linha na tela.

A tabela abaixo fornece os códigos hexadecimais destas instruções.

RST 0	C7
RST 8	CF
RST 10	D7
RST 18	DF
RST 20	E7
RST 28	EF
RST 30	F7
RST 38	FF

Por enquanto, nos interessa a instrução RST 10. Como vimos, *RST 10 coloca na tela o caractere cujo código está no acumulador*.

Assim, ela é equivalente à instrução BASIC da linha 20 do programa,

20 PRINT CHR\$ A;

É uma instrução poderosa e econômica: "custa" apenas um byte (D7)!

1.5

VOLTANDO AO BASIC

Já sabemos substituir o LET e o PRINT. Entretanto, ainda nos falta um detalhe importantíssimo. Ao contrário do BASIC, um programa ASSEMBLY não "põe" automaticamente na última instrução. É necessário "dizer" ao microprocessador que o programa acabou. Para isto, utilizaremos a instrução RET (mnemônico de RETURN).

Assim, não esqueça. Para finalizar o programa ASSEMBLY, use

RET

C9

Duas observações:

- 1^a) Se você esquecer de colocar o RET, o microprocessador interpretará o conteúdo do próximo endereço (que é totalmente imprevisível, e não 0, como se poderia supor) como uma instrução, e tentará executá-la. Assim sendo, *qualquer coisa pode acontecer!* Esta qualquer coisa chama-se CRASH.

O que é CRASH? A tela some, ou aparecem imagens completamente malucas (às vezes são parecidas com as linhas de LOAD e SAVE do BASIC, às vezes não). O teclado não funciona, nem a tecla BREAK. A única solução é desligar o computador (qualquer programa nele presente estará perdido), e ligá-lo novamente.

Apesar dos pesares, um consolo: nem o pior CRASH danifica o computador.

- 2^a) Talvez você saiba que existe em ASSEMBLY uma instrução equivalente ao STOP do BASIC: é a instrução HALT. JAMAIS a use em micros da linha SINCLAIR, pois HALT tira de você o acesso ao teclado, criando um estado semelhante ao CRASH: teclado desativado. Insistimos:

JAMAIS USE HALT (código hexa 76)

SEMPRE USE RET (código hexa C9)

1.6

O PRIMEIRO PROGRAMA ASSEMBLY

Nosso primeiro programa ASSEMBLY está pronto. Observe.

BASIC	ASSEMBLY	CÓDIGO HEXA
10 LET A = 23	LD A,23	3E 17
20 PRINT CHR\$ A;	RST 10 RET	D7 C9

1.7

O QUE FAZER COM ELE?

Agora, a pergunta que você deve estar se fazendo: como entrar com este programa no micro? Ele só "entende" BASIC!

É verdade. Se você digitar LDA,23 produzirá LET DA,23, o que lhe trará erro de sintaxe; se você digitar 3E 17 produzirá 3 REM 17, o que não funciona! Como introduzir um programa ASSEMBLY no computador?

Precisamos de um programa BASIC que faça isto para nós!

Primeiramente, reservaremos uma área do BASIC segura para colocar nela nosso programa ASSEMBLY, sem o risco de ele ser "coberto" pelo BASIC. Existem várias maneiras de fazer isto. A mais fácil (em nossa opinião) é usar uma linha inicial do programa de REM — ela será ignorada pelo BASIC!

Assim, a primeira providência para digitar um programa ASSEMBLY é digitar

1 REM tanta carateres (quaisquer, até espaços servem) quantos forem os bytes do programa.

Por exemplo, nosso programa ocupa 4 bytes (3E 17 D7 C9). Logo, é necessária uma linha 1 REM com pelo menos 4 carateres quaisquer. Pode ser

1 REM 1234 ou

1 REM VOVO ou

1 REM XXXX ou

1 REM VACA ou

1 REM qualquer coisa que você quiser que tenha pelo menos 4 carateres!

Vamos estudar isto com mais detalhes.

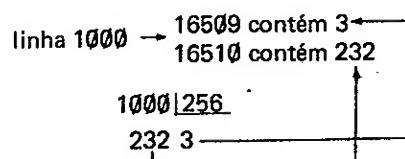
Suponha que você digitou 1 REM 1111. Na memória do micro, estará:

ENDEREÇO (d)	CONTEÚDO (d)	SIGNIFICADO
1º endereço BASIC 16509	0	número da linha: 0 x 256 + 1 = 1
(para maiores detalhes, consulte os Apêndices 5, 6 e 7).	16510	1
	16511	6
	16512	0
	16513	234
	16514	29
	16515	29
	16516	29
	16517	29
	16518	118

Vamos às explicações. Os endereços 16509 e 16510 guardam o número da linha, de maneira tal que 16509 guarda o byte mais significativo, e 16510 guarda o byte menos significativo. Os micros Sinclair só usam esta estrutura para guardar números de linha e variáveis de controle de loops FOR/NEXT. Consulte o seu manual!

Exemplos:

16509 contém 6d } linha número $6 \times 256 + 4 = 1540$
16510 contém 4d



Assim sendo, você pode transformar a linha 1 do seu programa em linha 0! (Você já deve ter visto isto em programas comerciais.) Basta digitar (modo direto)

POKE 16510,0

Digite NEW LINE, e após o 0/0, digite LIST. Lá está a sua linha 0. Se quiser que ela volte a ser 1, digite

POKE 16510,1

Brinque à vontade com isto, até ter certeza de ter entendido.

Vamos em frente. Os endereços 16511 e 16512 guardam o comprimento da linha, da maneira "normal" da Sinclair: O BYTE MENOS SIGNIFICATIVO VEM ANTES!

Nossa linha tem comprimento 6, observe:

REM 1111 NEW LINE

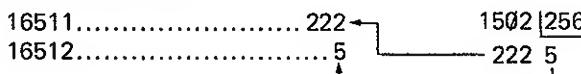
1 byte 4 bytes 1 byte

Uma linha 1 REM com 1500 caracteres quaisquer teria comprimento 1502!

REM 1500 caracteres quaisquer NEW LINE

1 byte 1500 bytes 1 byte

Os endereços conteriam:



Continuando: o endereço 16513 contém 234, que é o código decimal de REM. Se a linha fosse 1 PRINT, conteria 245, que é o código decimal de PRINT. Para todos os códigos de todos os comandos BASIC, veja o Apêndice 2.

Os endereços de 16514 a 16517 contêm 29, código decimal do caractere 1. Novamente, consulte o Apêndice 2 para qualquer informação adicional.

Finalmente: o endereço 16518 contém 118, que é o código de NEW LINE. Este NEW LINE faz parte da linha de BASIC!

Você pode, se quiser, fazer seu micro mostrar tudo isto que escrevemos para você. Digite o seguinte programa BASIC:

```
1 REM 1111
10 FOR E = 16509 TO 16518
20 PRINT E;"--->";PEEK E,CHR$ PEEK E;
30 NEXT E
```

Após o RUN, você deverá ter a seguinte saída de vídeo;

1 6 5 0 9 - - - > 0		R E M
1 6 5 1 0 - - - > 1		
1 6 5 1 1 - - - > 6		
1 6 5 1 2 - - - > 0		
1 6 5 1 3 - - - > 2 3 4		
1 6 5 1 4 - - - > 2 9		
1 6 5 1 5 - - - > 2 9		
1 6 5 1 6 - - - > 2 9		
1 6 5 1 7 - - - > 2 9		
1 6 5 1 8 - - - > 1 1 8		

Se você entendeu o que fizemos, sabe agora que podemos usar a nossa vontade o espaço de 16514 a 16517: está protegido numa linha de REM. Recomendamos que você digite sempre

POKE 16510,0

para transformar a linha 1 em linha 0. Isto defenderá você de apagá-la involuntariamente. (Eu próprio já apaguei, sem querer, uma linha que me custara quase *meia hora* de digitação, é frustrante.)

Agora estamos prontos para o programa BASIC que fará para nós a entrada de nosso programa ASSEMBLY.

1.8

O CARREGADOR ASSEMBLY

Primeiro, de que precisamos?

De um programa que faça POKE dos valores necessários nos endereços de 16514 a 16517. Para isto, o programa deverá converter os códigos de máquina hexadecimais em seus respectivos valores decimais. É fácil. Vamos lá. Digite NEW, 1 REM 4 caracteres quaisquer, e entre com o seguinte programa BASIC. Não esqueça do POKE 16510,0.

Segurança!

```
9000 FOR E=16514 TO 16517
9005 SCROLL
9010 PRINT E; "---->";
9015 INPUT H$
9020 POKE E,16 *CODE H$ + CODE H$(2) - 476
9025 PRINT H$
9030 NEXT E
```

Se você achar que vale a pena, grave este programa.

Digite RUN, e leia atentamente as observações abaixo.

- 1^{a)}) Naturalmente use os números de linha que desejar: 9000 é apenas uma sugestão.
- 2^{a)}) A linha 9000 dá os endereços inicial e final do nosso programa ASSEMBLY.
Logicamente deverá ser alterada para outros programas.
- 3^{a)}) A linha 9015 solicita que você digite os códigos hexa *UM DE CADA VEZ* (3E NEW LINE 17 NEW LINE D7 NEW LINE C9 NEW LINE).
- 4^{a)}) A linha 9020 converte hexa em decimal e faz o POKE adequado.

Exemplifiquemos com 3E.

$$16 * \text{CODE H\$} + \text{CODE H\$(2)} - 476 = 16 * \text{CODE 3} + \text{CODE E} - 476 = \\ = 16 * 31 + 42 - 476 = 496 + 42 - 476 = 538 - 476 = 62 \quad (\text{ver Apêndice 2})$$

Confira na tabela do Apêndice 3:

$$3Eh = 62d$$

Logo, o primeiro passo do loop faz

POKE 16514,62

e o primeiro caractere da linha de REM é alterado para Y! Veja o Apêndice 2!
E assim todos os caracteres que você tenha colocado na linha Ø serão alterados ...
Para outro carregador ASSEMBLY, veja o Apêndice 13.

1.9

DE VOLTA AO PRIMEIRO PROGRAMA ASSEMBLY

Nosso primeiro programa ASSEMBLY já está no computador. E agora, como rodá-lo?
RUN é agora inútil, pois faz rodar o programa BASIC ...
Antes de ver como rodar o programa, acostume-se à forma como escreveremos nossos
programas nesta fase inicial da aprendizagem:

PROGRAMA Ø1	(4 bytes)
-------------	-----------

Ø REM 4 caracteres quaisquer

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	3E	LD A,23
16515	17	RST 10
16516	D7	
16517	C9	RET

Vamos agora conferir o que fizemos.

Dê um LIST e observe atentamente: sua linha Ø deve apresentar o seguinte aspecto:

Ø REM Y* NOT TAN

Se não tiver estes caracteres, digite RUN outra vez e entre CUIDADOSAMENTE com os
códigos hexa. Qualquer erro estraga tudo. *Em ASSEMBLY praticamente não há pequenos
erros: quase todos eles levam ao CRASH.*

Uma vez obtida a linha

Ø REM Y* NOT TAN

estude os códigos hexa do programa Ø1, compare com os caracteres da tabela do
Apêndice 2, e você entenderá como surgiu esta estranha linha REM.

Mas ... agora o que queremos é rodar o programa! A função do BASIC que dá acesso a
rotinas ASSEMBLY é USR (mnemônico para USE ROUTINE). Mas é impossível digitá-la
diretamente, pois é função, e não comando. Assim, colocaremos antes dela a palavra-chave
(comando) RAND que, no nosso caso, não faz absolutamente nada! Apenas permite a
digitação de USR sem erro de sintaxe.

Até que enfim, vamos rodar. Limpe a tela (CLS). Digite (modo direto)

RAND USR 16514

Lá está o seu asterisco no canto superior esquerdo da tela! RAND USR 16514 também pode ser linha de programa BASIC. Faça:

```
Ø REM Y* NOT TAN  
1 RAND USR 16514  
2 STOP
```

Digite RUN. O asterisco aparece no canto superior esquerdo da tela, e o programa pára com código 9/2.

A linha 2 STOP é necessária para não invadirmos o programa carregador, que continua lá na linha 9000.

Observe que, apesar de sua incrível simplicidade, este programa é um mistério para quem só conhece BASIC. Veja os "mistérios":

- a) A linha Ø;
- b) Qual a finalidade da linha REM? Linhas REM não são "ignoradas" pelo computador?
- c) RAND? Isto é para obter números randômicos ...
- d) USR? O que é isto?
- e) 16514? Que número é este?
- f) Como é que aquele asterisco foi parar ali?

E, para nós, que já nos iniciamos na poderosa linguagem ASSEMBLY, estes mistérios estão claríssimos ...

(Se algum destes "mistérios" está realmente misterioso para você, volte atrás *agora* e estude novamente o capítulo até aqui. Tudo o que vimos é fundamental.)

1:10

ALTERANDO O CARACTERE

Se você entendeu bem o programa, sabe que o endereço 16515 contém agora 23, código decimal do caractere *. Logo, é possível alterar o caractere impresso pelo programa, fazendo um POKE neste endereço.

Lembre-se que o comando POKE trabalha com numeração *decimal*. Assim sendo, digite (modo direto):

```
POKE 16515,151  
RAND USR 16514
```

Você deve ter agora um * no canto superior esquerdo do vídeo.

Um cuidado especialíssimo: se o acumulador A for carregado com algum valor que NÃO seja o código de um caractere imprimível, a instrução RST 10 provocará CRASH do sistema.

Consulte a tabela do Apêndice 2 e comprove que os caracteres imprimíveis têm códigos decimais x tais que

$0 \leq x \leq 63$ (em hexa $0 \leq x \leq 3F$)	ou então	$128 \leq x \leq 191$ $.80 \leq x \leq BF$
---	----------	---

Mais tarde veremos que tais números guardam uma incrível correspondência, que nos permitirá INVERTER O VÍDEO! Aguarde o Capítulo 8.

Se, ao fazer POKE 16510,x; o valor x estiver fora de uma das faixas especificadas, CRASH ...

1.11

EXERCÍCIOS

- 05 Se possível sem "colar" do PROGRAMA 01, mas consultando os apêndices que julgar convenientes, complete as lacunas abaixo, escrevendo um programa que coloque um sinal de + no canto superior esquerdo da tela.

Ø REM ... caracteres quaisquer

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	LD,
.....	RST
.....
.....

- 06 Sobre o programa que você fez, responda:

- Para rodá-lo, devemos digitar
- Qual o aspecto da linha Ø REM?
- Para que a saída de vídeo seja um sinal de + em vídeo inverso, devemos digitar POKE,
- Se digitarmos POKE 16515,216
RAND USR 16514
ocorrerá porque 216

1.12

CHAMANDO ROTINAS NA ROM

Bem, desligue o micro, e vamos estudar.

A instrução RST 10 é a maneira mais econômica de imprimir um caractere no vídeo (apenas um byte).

No entanto, vamos ver outras maneiras, que têm cada qual a sua utilidade.

Existem duas rotinas na ROM que também imprimem no vídeo. Seus endereços são

HEXADECIMAL DECIMAL

Ø7 F1	2033 ($7 \times 256 + 241$)
Ø8 Ø8	2056 ($8 \times 256 + 8$)

Em BASIC, necessitaremos da instrução GOSUB. Em ASSEMBLY, usaremos CALL (código hexadecimal: CD).

CALL	CD
------	----

Ao escrever o endereço vamos usar 2 bytes, e você deve colocar o byte menos significativo primeiro.

MUITO CUIDADO! Observe:

CALL 2056 CD 08 08
CALL 2033 CD F1 07 (e NÃO CD 07 F1)

CD 07 F1 chama uma sub-rotina no endereço 61703, só existente se você tiver uma expansão de 64 K! Logo, muito cuidado!

Conhecendo agora a instrução CALL, o programa 01 pode ser substituído pelo

PROGRAMA 02 (6 bytes)

Ø REM 6 caracteres quaisquer

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	3E	{ LD A,23
16515	17	
16516	CD	{ CALL 2056
16517	08	
16518	08	
16519	C9	RET

Em relação ao programa 01, este apresenta as seguintes desvantagens:

- 1º) Gasta 2 bytes a mais;
- 2º) A instrução CALL 2056 *destrói* os registros. Depois dela, o acumulador NÃO mais está carregado com o código de * (na verdade, é imprevisível o conteúdo do acumulador). Isto pode ser uma desvantagem séria, mas ... aprenderemos a contorná-la. Há recursos de sobra!

1.13

EXERCÍCIOS

- Ø7 Complete as lacunas, escrevendo um programa equivalente ao 02, que coloque uma interrogação no canto superior esquerdo da tela, usando a sub-rotina do endereço 2033d da ROM.

Ø REM ... caracteres quaisquer

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	LD.....	{ LD.....
.....	
.....	{ CALL 2033
.....	
.....

- 08 Observe atentamente o programa abaixo. Pela primeira vez, você vai fazer a desmontagem (DISASSEMBLY) de um programa. Este exercício será freqüente para nós: é muito importante. É este trabalho que permitirá a você entender os programas escritos por outros ... Complete as lacunas, e depois responda às perguntas:

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	3E	
16515	17	}
16516	D7	
16517	D7	
16518	D7	
16519	C9	

- a) Para carregá-lo no computador, é necessária uma linha Ø REM de quantos caracteres?
- b) Por quê?
- c) O que faz o programa? Responda a isto SEM rodá-lo. Depois rode e confira.
- d) Seria possível a mesma estrutura usando CALL 2Ø33? Reporte-se à segunda observação do item 1.12.

RESPOSTAS DOS EXERCÍCIOS

- 01 a) 240
b) 15
c) ØØ (observe que números hexadecimais são escritos — para nosso uso — com dois dígitos)
d) 79
e) 57
f) 255
- 02 a) LD A,151 3E 97
b) LD B,32 Ø6 2Ø
c) LD L,16 2E 1Ø
d) LD C,24Ø ØE FØ
e) LD H,255 26 FF
f) LD E,128 1E 8Ø
g) LD D,8 16 Ø8
- 03 Não, pois 7Ø4 > 255. O registrador B só pode conter um byte, ou seja, valores decimais de Ø até 255.
- 04 a) A, B, C, D; E, H e L
b) 255; FF
c) LOAD, carregar
d) acumulador
e) LD A,255

05 Ø REM 4 caracteres quaisquer

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	3E	
16515	15	}
16516	D7	RST 10
16517	C9	RET

06 a) RAND USR 16514

b) Ø REM Y+NOT TAN

c) POKE 16515,149

d) Ocorrerá CRASH, porque 216 não é o código de um caractere imprimível
(** NÃO É UM CARACTERE).

07 Ø REM 6 caracteres quaisquer

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	3E	
16515	0F	}
16516	CD	
16517	F1	}
16518	07	CALL 2033
16519	C9	RET

ENDEREÇO	CÓDIGO HEXA	INSTRUÇÃO
16514	3E	
16515	17	}
16516	D7	RST 10
16517	D7	RST 10
16518	D7	RST 10
16519	C9	RET

a) Ø REM 6 caracteres quaisquer.

b) Porque o programa ocupa 6 bytes.

c) Coloca 3 * seguidos na tela.

d) Não, pois CALL 2033 destrói os registros. Aprenderemos mais tarde como contornar este problema. Como já dissemos, há recursos!

2.1

INTRODUÇÃO

Para encher a tela com um caractere, o seguinte programa BASIC seria possível:

```
10 LET A = 23  
20 PRINT CHR$ A;  
30 GOTO 20
```

Após o RUN, a tela leva alguns segundos (cerca de 10) para ficar repleta de asteriscos e aparecer o código 5/20.

Como você já sabe, 5 é o código de falta de espaço na tela. Digitando-se CONT, a tela é limpa, e novamente é preenchida.

Observando o programa, sabemos como substituir em ASSEMBLY as linhas 10 e 20. O que nos falta é algo equivalente a GOTO. É o que veremos neste capítulo.

2.2

SALTO ABSOLUTO INCONDICIONAL

A instrução JUMP (mnemônico JP) seguida de 2 bytes para o endereço é equivalente a GOTO em BASIC. Seu código hexadecimal é

JP	C3
----	----

Da mesma forma que CALL, o endereço a seguir deve ser escrito com o byte *menos* significativo primeiro.

Assim, se desejarmos fazer JP 16514 devemos fazer

16514	1256
1154	64
130	40 em hexa, este é o byte MAIS significativo
82	82 em hexa, este é o byte MENOS significativo

Logo, a instrução JP 16514 fica C3 82 40

primeiro o byte
MENOS
significativo

2.3

ENCHENDO A TELA

Você vai ser apresentado a uma das maiores vantagens da linguagem ASSEMBLY:
a *velocidade*.

PROGRAMA #3

(6 bytes)

Ø REM 6 caracteres quaisquer

16514	3E	}	LD A,23
16515	17		RST 10
16516	D7	}	JP 16516
16517	C3		
16518	84		
16519	40		

Observe bem: a instrução JP 16516 substitui o GOTO 20 do programa BASIC.

Digite RAND USR 16514 e veja a incrível velocidade com que a tela é preenchida (menos de 1 segundo!)

Velocidade é a segunda principal vantagem do uso de ASSEMBLY. A principal é economia de memória. Isto você só perceberá mais tarde.

2.4

EXERCÍCIOS

09 Preencha as lacunas:

- a) JP é o mnemônico para , que significa
É equivalente à instrução BASIC
- b) O código hexa da instrução JP é

10 Você vai preparar um programa que enche a tela com a letra H. Siga o esquema.

INÍCIO → LD A,N (N = código do caractere H)
CALL 2056
JP INÍCIO

À palavra INÍCIO, usada para determinar um ponto do programa, chamamos de um RÓTULO (LABEL). Seu uso é extremamente útil.

Complete então:

Ø REM caracteres quaisquer

INÍCIO → 16514 }
16515 }
16516 }
16517 }
16518 }

16519	}
16520
16521

Após rodar o programa, através do comando , pense e justifique por que o rótulo INÍCIO tem de ser em 16514, e não em 16516, como no programa Ø3. Isto é porque a instrução destrói

2.5

SALTO RELATIVO INCONDICIONAL

Existe outra instrução de salto, que não tem equivalência direta com o BASIC, mas que nos será extremamente útil. É a instrução JUMP RELATIVE (mnemônico JR), seguida de 1 byte, que dá a magnitude do salto. Seu código hexadecimal é

JR	18
----	----

O byte seguinte ao endereço dará a magnitude do salto, para frente ou para trás, baseado na seguinte convenção:

entre 00 e 7F → salto para frente entre 80 e FF → salto para trás
--

Estudemos separadamente os casos:

a) SALTO PARA FRENTE

JR Ø não tem sentido, é uma instrução para não saltar! JR n diz que você deve saltar n endereços a partir do final da instrução. Observe o exemplo:

16514	18	JR 2
16515	02	
16516	XX	estes 2 endereços serão saltados, o programa salta para o endereço 16518
16517	XX	
16518	

O salto mais amplo é JR 127 (7Fh = 127d). Observe o exemplo:

16514	18	JR 127
16515	7F	
16516	estes 127 endereços serão saltados, o programa salta para o endereço $16516 + 127 = 16643$
16517	
..	..	
..	..	
..	..	
16642	
16643	

b) SALTO PARA TRÁS

O salto para trás é baseado numa convenção de números negativos, chamada COMPLEMENTO DE DOIS. Esta tabela está no Apêndice 4.

Ao calcular o número de bytes saltados para trás, devemos incluir os 2 bytes gastos pela própria instrução. Observe o exemplo:

17000	
17001	
17002	
17003	18 }	JR -4
17004	FC	

estes 4 endereços estão envolvidos no salto, será realizada a instrução do endereço 17001.

Observe agora um detalhe muito importante: você NÃO PODE fazer JR -1 ou JR -2. JR -2 (18 FE) é um loop infinito, e o único recurso será desligar o computador.

Faremos agora um programa para exemplificar o salto relativo para trás. Atenção, pois é o programa mais curto que enche a tela.

PROGRAMA 04

1 REM 5 caracteres quaisquer

16514	3E }	LD A,23
16515	17	
16516	D7	RST 10
16517	18 }	JR -3
16518	FD	

Após a instrução JR -3, o programa volta ao endereço 16516, ou seja, à instrução RST 10.

JR -5(18 FB) também funcionaria, mas não há necessidade de recarregar o acumulador, pois RST 10 NÃO destrói os registros.

A instrução JR é mais econômica do que JP (2 bytes x 3 bytes), mas necessita mais cuidados para ser usada: é necessário contar cuidadosamente o número de bytes a serem saltados, principalmente nos saltos para trás.

Nos saltos para trás, não esqueça de incluir os 2 bytes da própria instrução.

2.6

EXERCÍCIOS

11 Faça a desmontagem do programa abaixo. Depois responda às perguntas.

16514	3E
16515	17	
16516	CD	
16517	F1 }
16518	07	
16519	18 }
16520	F9	

- a) É necessária uma linha \emptyset REM com quantos caracteres?
- b) Por que o endereço 16520 contém F9 e não FB?
- c) Aproveitando a linha \emptyset REM, faça:
- 10 RAND USR 16514
 20 CLS
- Digite RUN e responda: a linha 20 funciona? Por quê?
 Acrescentar 16521 C9 (RET) resolveria o problema?
- 12 Que aconteceria com um programa que contivesse a instrução
 18 FE (JR - 2)?

- 13 Um desafio. Vamos escrever um programa BASIC, e você deverá convertê-lo para ASSEMBLY.

10 PRINT A;
 20 PRINT D;
 30 GOTO 10

Naturalmente este programa enche a tela com as letras A e D. Sugerimos a seguinte estrutura:

INÍCIO —→ LD A,código do caractere A
 RST 10
 LD A,código do caractere D
 RST 10
 JR INÍCIO

Claro que JP INÍCIO também serviria, mas JR é mais econômico.

RESPOSTAS DOS EXERCÍCIOS

- 09 a) JUMP, salto, GOTO
 b) C3
- 10 \emptyset REM 8 caracteres quaisquer.

INÍCIO —→ 16514 3E } LD A,45
 16515 2D }
 16516 CD }
 16517 08 } CALL 2056
 16518 08 }
 16519 C3 }
 16520 82 } JP 16514
 16521 40 }

RAND USR 16514; CALL 2056; todos os registros.

- 11 16514 3E } LD A,23
 16515 17 }
 16516 CD }
 16517 F1 } CALL 2033
 16518 07 }

16519 18 } JR -7
16520 F9 }

- a) É necessária uma linha Ø REM com 7 caracteres quaisquer, pois o programa ocupa 7 bytes.
- b) F9, na convenção de complemento de dois, corresponde a -7; o que leva o programa de volta ao endereço 16514.
FB corresponde a -5, o que levaria o programa ao endereço 16516, ou seja, CALL 2033 sem o registrador A estar carregado (pois já teriam sido destruídos os registros pela própria instrução CALL 2033). Falando com simplicidade: CRASH.
- c) Não, não funciona, da mesma maneira que não funciona a linha 4Ø do programa BASIC abaixo:

```
10 LET A = 23
20 PRINT CHR$ A;
30 GOTO 10
40 CLS
```

Acrescentar 16521 C9 (RET) não resolve o problema, pois o programa não chegará jamais a este endereço!

Mais tarde aprenderemos a contornar este problema.

- 12 O programa entraria em loop infinito, pois a instrução 18 FE (JR -2) volta ao início dela mesma.
- 13 Ø REM 8 caracteres quaisquer

INÍCIO —————> 16514 3E } LD A,38
 16515 26 }
 16516 D7 RST 1Ø
 16517 3E } LD A,41
 16518 29 }
 16519 D7 RST 1Ø
 16520 18 } JR -8
 16521 F8 }

3.1

INTRODUÇÃO

Escrever uma string usando RST 10 torna o programa longo, pois é necessário carregar o acumulador e chamar RST 10 para cada caractere. Como exemplo da maneira como você NÃO deve usar para escrever strings na tela, observe atentamente o programa abaixo.

PROGRAMA 05

(13 bytes)

Ø REM 13 caracteres quaisquer

16514	3E	26	LD A,38
16516	D7		RST 10
16517	3E	36	LD A,50
16519	D7		RST 10
16520	3E	34	LD A,52
16522	D7		RST 10
16523	3E	37	LD A,55
16525	D7		RST 10
16526	C9		RET

Você deve ter reparado que escrevemos as instruções de maneira um pouco mais compacta. Os bytes que formam uma instrução são escritos em seqüência *horizontal* e, quanto aos endereços, é dado apenas o endereço *inicial* de cada instrução.

Sobre o programa em si, vamos agora a alguns ...

3.2

EXERCÍCIOS

14 Sem rodar o programa, responda:

a) O que ele faz?

.....

b) Se, aproveitando a linha Ø, fizermos

10 RAND USR 16514

20 CLS

30 GOTO 10

funciona? Em caso afirmativo, o que faz?

.....

c) Modifique o final do programa 05 para que ele encha a tela com a string que ele escreve. Duas soluções são possíveis: uma usando um salto absoluto para 16514, outra usando salto relativo. Escreva ambas as soluções.

1^a) Com salto absoluto:

1652... JP } Agora o programa tem bytes,
a linha 0 REM tem de ser aumentada.

2^a) Com salto relativo:

1652... JR } Agora o programa tem bytes,
a linha 0 REM tem de ser aumentada.

d) Se, aproveitando a linha 0 do programa MODIFICADO POR VOCÊ como sugerido no item c, fizermos

10 RAND USR 16514
20 CLS
30 GOTO 10

funciona? Em caso afirmativo, o que faz?
.

15 Quantos bytes seriam necessários para escrever o alfabeto completo na tela, usando a técnica do programa 05? bytes.

3.3

PARES DE REGISTRADORES

A resposta do exercício 15 mostra claramente a impossibilidade prática de usarmos RST 10 para escrever uma string longa. Logo, é necessário um outro processo. Existem vários, todos eles usando pares de registradores. Se achar que vale a pena, e se já se julgar em condições de entender, leia o Apêndice 1.

Por enquanto, o que nos interessa é que registradores de 8 bits podem se juntar, formando pares de registradores que aceitam um carregamento de 16 bits.

Podemos carregar diretamente os pares de registradores BC, DE e HL com números entre 0 e 65535 (0000 até FFFF).

B, D e H conterão o byte MAIS significativo; C, E e L conterão o byte MENOS significativo.

Se você fala inglês, gostará desta observação: o par HL usa estas letras devido a High Low (alto baixo), para lembrar ao usuário que H tem a parte alta (High) do número (o byte mais significativo), e que L tem a parte baixa (Low) do número (o byte menos significativo).

Assim, para que o par HL contenha o valor 704 decimal, é necessário que L contenha 192d (C0h) e que H contenha 2d (02h). Observe:

704	256
192	2
↓ ↓	
C0h, conteúdo de L	02h, conteúdo de H

Como já aconteceu com CALL e com JP, ao escrever a instrução de carregamento de um par de registradores, primeiramente o byte *menos* significativo, depois o byte *mais* significativo.

Vamos agora aprender a carregar (LOAD) estes pares de registradores. A instrução tem o aspecto geral

LD rr,NN

onde

LD → mnemônico de LOAD

rr → par de registradores (BC, DE ou HL)

NN → número entre 0 e 65535d (0000 e FFFFh)

Novamente, a correspondência com BASIC é simples:

BASIC	ASSEMBLY
LD HL,1000	LET HL = 1000
LD BC,704	LET BC = 704

A instrução custa 3 bytes: um para a instrução propriamente dita e dois para o número.

A tabela abaixo fornece os códigos hexa destas instruções.

LD BC,NN	01.....
LD DE,NN	11.....
LD HL,NN	21.....

3.4

EXERCÍCIOS

16 Complete as lacunas:

INSTRUÇÃO	CÓDIGO HEXA
a) LD DE,.....82 40 (não esqueça da inversão!)
b) LD BC,.....37 00
c) LD,36	01.....
d) LD,.....	21 0B 6B

17 Muito cuidado com a inversão! Complete as lacunas:

a) LD BC,.....25 35
b) LD BC,.....35 25

3.5

COLOCANDO UMA STRING NA TELA

Provavelmente o processo mais fácil de colocar uma string na tela seja chamando (CALL) a sub-rotina 2923 (0B 6B em hexa). Para chamá-la, é necessário que o par de registradores

DE contém o endereço de início da string, e o par BC contém o número de caracteres da string (LEN). Se você fala inglês, outra vez uma observação útil: o par BC é o Byte Counter (contador de bytes).

A estrutura de montagem do programa é a seguinte:

- Digitase a string no início da linha de REM. Isto permite-nos saber o endereço do início da string, que deve ser carregado em DE: 16514.
- Conta-se *cuidadosamente* quantos caracteres compõem a string, sem esquecer dos espaços. Este número será carregado em BC.

Como exemplo, vamos colocar na tela a string LINGUAGEM DE MAQUINA. Conte: esta string tem 20 caracteres (incluindo os dois espaços). Logo, ela ocupará os endereços de 16514 até 16533 ($16533 = 16514 + 20 - 1$). Enfatizando: *a string deve ser colocada no início da linha de REM*.

Assim sendo, o primeiro endereço de *programa* (não de dados) será 16534.

Após a digitação da string, precisamos reservar espaço para o programa. Vamos calcular quantos bytes ele gastará.

- Cálculo do número de bytes de programa:
 - carga do par de registradores DE com o endereço inicial da string (16514):
3 bytes → LD DE, 16514 → 11 82 40
 - carga do par de registradores BC com o número de caracteres a serem impressos (20):
3 bytes → LD BC, 20 → 01 14 00
 - chamada da sub-rotina 2923 da ROM, que imprime strings:
3 bytes → CALL 2923 → CD 6B 0B
 - retorno ao BASIC:
1 byte → RET → C9

Agora estamos prontos.

PROGRAMA 06

(20 bytes + 10 bytes)

0 REM LINGUAGEM DE MAQUINA e mais 10 caracteres quaisquer.
(endereços de 16514 a 16533 ocupados pela string)

16534	11 82 40	LD DE, 16514
16537	01 14 00	LD BC, 20
14540	CD 6B 0B	CALL 2923
16543	C9	RET

Certos cuidados devem ser tomados na entrada do programa no computador:

- o endereço inicial é 16534, e não 16514 como de hábito;
- o endereço final é 16543, o endereço da famosa instrução RET;
- assim sendo, altere convenientemente a linha 9000 do programa de carga que demos no item 1.8;

- para rodar o programa, digite RAND USR 16534 (e não RAND USR 16514 como de costume);
- este último cuidado é vital, senão... CRASH!

3.6

EXERCÍCIOS

Estes exercícios são muito importantes. Não passe adiante sem resolvê-los.

- 18 Faça um programa que escreva o alfabeto completo na tela. (Mais tarde, faremos um programa bastante mais curto que faz isto também.) Calcule cuidadosamente os endereços, isto é crítico.
- 19 Prepare um programa que encha a tela com a string

* EDITORA * CAMPUS *

Como você pode contar, esta string tem 16 caracteres; logo, "cabe" 2 vezes em cada linha, fazendo um belo efeito de vídeo.

Substitua a instrução RET por um salto relativo que leve ao *primeiro byte de programa*, não de dados.

Muito cuidado com os endereços e com o salto para trás. Não esqueça de consultar a tabela de complemento de dois.

Bom trabalho ...

RESPOSTAS DOS EXERCÍCIOS

- 14 a) Imprime na tela a string AMOR.
b) Sim, funciona. Faz a palavra AMOR piscar no canto superior esquerdo da tela.
c) Primeiro que tudo esperamos que você tenha lembrado de *retirar* 16526 C9 RET, do contrário... não funcionará!
1^a) Com salto absoluto:
16526 C3 82 40 JP 16514
Agora o programa tem 15 bytes, logo não esqueça de aumentar a linha
Ø REM!
2^a) Com salto relativo:
16526 18 F2 JR -14
Agora o programa tem 14 bytes.
d) Não, não funciona. O programa *não* atinge a linha Ø. Pára com código 5/10.
15. $3 \times 26 + 1 = 79$ bytes
3 → para cada letra, 3 bytes (2 para carga do acumulador e 1 para RST Ø)
26 → 26 letras no alfabeto
1 → 1 byte para RET

- 16 a) LD DE,16514 11 82 40
 b) LD BC,55 01 37 00
 c) LD BC,36 01 24 00
 d) LD HL,27403 21 0B 6B
- 17 a) LD BC,13605 01 25 35 ($53 \times 256 + 37 = 13605$)
 b) LD BC,9525 01 35 25 ($37 \times 256 + 53 = 9525$)
- 18 BC deve ser carregado com 26 (26 letras do alfabeto).
 DE deve ser carregado com 16514 (endereço inicial da string). A string ocupa os endereços de 16514 até $16514 + 26 - 1 = 16539$.
 Logo, o primeiro endereço de programa é 16540. Como o programa gasta 10 bytes, o endereço final é $16540 + 10 - 1 = 16549$.

Ø REM ABCDEFGHIJKLMNOPQRSTUVWXYZ e mais 10 caracteres quaisquer.

16540	11	82	40	LD DE,16514
16543	01	1A	00	LD BC,26
16546	CD	6B	0B	CALL 2923
16549	C9			RET // RAND USR 16540

- 19 BC deve ser carregado com 16 (número de caracteres da string).
 DE deve ser carregado com 16514 (endereço inicial da string).
 A string ocupa os endereços de 16514 até $16514 + 16 - 1 = 16529$.
 Logo, o primeiro endereço do programa é 16530.
 Calculamos o número de bytes do programa, para sabermos a magnitude do salto relativo para trás.

LD BC,16	→ 3 bytes
LD DE,16514	→ 3 bytes
CALL 2923	→ 3 bytes
JR	→ 2 bytes
TOTAL	→ 11 bytes

Logo, precisamos de JR -11. Vamos ao programa.

Ø REM * EDITORA * CAMPUS * e mais 11 caracteres quaisquer.

16530	11	82	40	LD DE,16514
16533	01	10	00	LD BC,16
16536	CD	6B	0B	CALL 2923
16539	18	F5		JR -11 // RAND USR 16530

4.1

A SUB-ROTINA PRINT AT

Existe na ROM uma sub-rotina que testa os parâmetros do comando PRINT AT (linha,coluna) quando este é usado a partir do BASIC. Seu endereço é 2293 (08F5 em hexa), e ela pode ser chamada (CALL) em nossos programas em ASSEMBLY.

A sub-rotina usa os registradores B e C, sendo que o número da linha (de 0 a 21) deve ser colocado no registrador B, e o número da coluna (de 0 a 31) é colocado no registrador C.

Vamos usar um truque: se fizermos LD B,linha e LD C,coluna, gastaremos 4 bytes. Assim, usaremos LD BC,....., o que gasta 3 bytes. Apenas é necessário tomar cuidado com a inversão!

Devemos carregar BC e chamar a sub-rotina ANTES de carregar o acumulador A com o código do caractere a ser impresso, pois esta sub-rotina (como todas as outras que já vimos) "limpa" todos os registros.

O efeito de CALL 2293 é equivalente à instrução BASIC

PRINT AT linha,coluna;

pois posiciona o cursor de impressão (invisível).

Como exemplo, vamos escrever um programa que coloque um asterisco *no meio* da tela (em 11,16).

PROGRAMA 07

(10 bytes)

0 REM 10 caracteres quaisquer

16514	01	10	0B	LD BC,2832 (B = 11; C = 16)
16517	CD	F5	08	CALL 2293
16520	3E	17		LD A,23
16522	D7			RST 10
16523	C9			RET

Algumas observações se fazem necessárias.

- 1º) O número 2832 não tem nenhum significado para nós. Estamos apenas economizando memória. Se fizéssemos LD B,11 (em B a linha) e LD C,16 (em C a coluna), gastaríamos 4 bytes. Fazendo LD BC,2832 (2832 = 11 x 256 + 16, onde 11 = linha = conteúdo de B[0Bh] e 16 = coluna = conteúdo de C[10h]), gastamos apenas 3 bytes. Não calcularemos mais este número. Para quê?
- 2º) Todo o cuidado é pouco para não esquecer da inversão (*o conteúdo de C é colocado primeiro*).

- 3ª) A instrução LD A,23 não pode ser colocada no início do programa, pois CALL 2293 a destruiria.

4.2

EXERCÍCIOS

- 20 Elabore um programa que coloque um ponto na última posição da tela, ou seja, em 21,31.
- 21 Prepare um programa que encha a *metade inferior* da tela com o caractere 08. Deve ter a mesma estrutura que o programa 07, substituindo-se o RET por um salto relativo para trás.

4.3

COMBINANDO SUB-ROTIAS DA ROM

No programa 06 colocamos na tela a string LÍNGUAGEM DE MAQUINA, através da sub-rotina 2923 (0B6B). Vamos combinar este programa com a chamada de 2293 (08F5) para centrá-la na tela.

PROGRAMA 08

(20 bytes + 16 bytes)

0 REM LÍNGUAGEM DE MAQUINA e mais 16 caracteres quaisquer

16534	01	06	08	LD BC,XXXX (B = 11; C = 6)
16537	CD	F5	08	CALL 2293
16540	11	82	40	LD DE,16514
16543	01	14	00	LD BC,20
16546	CD	6B	0B	CALL 2923
16549	C9			RET

Observe que fizemos B = 11 e C = 6, para posicionar o cursor invisível de impressão em 11,6. Naturalmente isto tem de ser a primeira parte do programa, pois CALL 2293 destrói o conteúdo dos registradores.

Para rodar o programa, você lembra: RAND USR 16534.

4.4

EXERCÍCIOS

Novamente, exercícios importantes. Não passe adiante sem resolvê-los. É necessário que você se acostume a calcular endereços e saltos para trás. Faça todo o possível — você é capaz — para resolvê-los sem “colar” as respostas.

- 22 Escreva um programa que coloque a palavra COMPUTADOR no final da sétima linha da tela.
- 23 Escreva um programa que coloque a string *FIM* no final da última linha da tela.

- 24 Escreva um programa que encha as 7 linhas *inferiores* da tela com a string *MICRO*SINCLAIR*. Como você pode contar, esta string tem 16 caracteres, logo "cabe" duas vezes em cada linha. Raciocine cuidadosamente para qual endereço você vai dirigir seu salto para trás.

RESPOSTAS DOS EXERCÍCIOS

- 20 Ø REM 10 caracteres quaisquer

16514	Ø1	1F	15	LD BC,XXXX (B = 21; C = 21)
16517	CD	F5	Ø8	CALL 2293.
1652Ø	3E	1B		LD A,27
16522	D7			RST 1Ø
16523	C9			RET

- 21 Metade superior da tela: 11 linhas, de Ø a 10

Metade inferior da tela: 11 linhas, de 11 a 21

Assim sendo, a primeira posição de impressão tem de ser 11, Ø.

- Ø REM 11 caracteres quaisquer

16514	Ø1	ØØ	ØB	LD BC,XXXX (B = 11, C = Ø)
16517	CD	F5	Ø8	CALL 2293
1652Ø	3E	Ø8		LD A,8
16522	D7			RST 1Ø
16523	18	FD		JR -3

Se você fez 18 FB (JR -5) também serve, embora seja desnecessário recarregar o acumulador, porque RST 1Ø NÃO destrói os registros.

Qualquer salto diferente destes não funciona. Raciocine e conclua por quê.
Deixamos este trabalho para você.

- 22 Antes de mais nada, sétima linha é a de número 6. Como a palavra COMPUTADOR tem 10 caracteres, ocupa da posição 22 até a posição 31. Logo, o comando BASIC seria

PRINT AT 6,22;"COMPUTADOR"

- Ø REM COMPUTADOR e mais 16 caracteres quaisquer

16524	Ø1	16	Ø6	LD BC,XXXX (B = 6; C = 22)
16527	CD	F5	Ø8	CALL 2293
1653Ø	11	82	ØØ	LD DE,16514
16533	Ø1	ØA	ØØ	LD BC,1Ø
16536	CD	6B	ØB	CALL 2923
16539	C9			RET // RAND USR 16524

- 23 O programa é equivalente ao comando BASIC

PRINT AT 21,27;"*FIM**"

- Ø REM *FIM* e mais 16 caracteres quaisquer
(endereços de 16514 a 16518 ocupados pela string)

16519	01	1B	15	LD BC,XXXX (B = 21; C = 27)
16522	CD	F5	08	CALL 2293
16525	11	82	40	LD DE,16514
16528	01	05	00	LD BC,5
16531	CD	6B	0B	CALL 2923
16534	C9			RET // RAND USR 16519

- 24 Primeiramente, as 7 linhas inferiores da tela são as linhas de números 15 a 21. Logo, a primeira posição de impressão é 15,0.

Ø REM *MICRO*SINCLAIR* e mais 17 caracteres quaisquer
(endereços de 16514 até 16529 ocupados pela string)

16530	01	00	0F	LD BC,XXXX (B = 15, C = Ø)
16533	CD	F5	08	CALL 2293
16534	11	82	40	LD DE,16514
16537	01	10	00	LD BC,16
16540	CD	6B	0B	CALL 2923
16543	18	F5		JR -11 // RAND USR 1653Ø

Observe que o salto relativo para trás deve voltar ao trabalho de colocação da string na tela, e não ao posicionamento do cursor. Caso contrário, não sairemos da primeira vez em que a string é impressa.

5.1

INTRODUÇÃO

Neste capítulo vamos aprender a fazer loops em linguagem ASSEMBLY, ou seja, a substituir a instrução FOR/NEXT do BASIC.

Porém, é necessário enriquecer o conjunto de instruções que conhecemos, para que possamos ter maior flexibilidade. Recapitulando — sabemos apenas 8 instruções:

- | | |
|-----------------------|--------------|
| a) LD r,N | (Capítulo 1) |
| b) LD rr,NN | (Capítulo 3) |
| c) RST 10 | (Capítulo 1) |
| d) CALL incondicional | (Capítulo 1) |
| e) JP incondicional | (Capítulo 2) |
| f) JR incondicional | (Capítulo 2) |
| g) RET | (Capítulo 1) |
| h) HALT | (NÃO USE!) |

Isto é muito pouco. A primeira coisa que precisamos ver melhor é como trabalha a função USR.

5.2

A FUNÇÃO USR

USR é o mnemônico de USE ROUTINE, ou seja, programas ASSEMBLY são considerados como sub-rotinas do BASIC. Assim, um programa BASIC

```
.....  
50.....USRxxxxx  
60.....  
.....
```

ao atingir a linha 50 executaria a sub-rotina do endereço xxxx e voltaria (ao encontrar RET) para executar a linha 60.

Mas... USR é uma função, não um comando. Assim, necessita um comando antes, do contrário você obtém erro de sintaxe. Temos usado RAND USR xxxx, onde RAND não "faz" absolutamente nada, apenas impede o erro de sintaxe.

O comando LET também serviria, da seguinte maneira:

LET BRUXA = USRxxxx,

onde BRUXA substitui qualquer nome válido para uma variável, e xxxx é o endereço inicial do nosso programa ASSEMBLY.

Ao encontrar a função USR, o computador roda o programa ASSEMBLY, porém desta vez com uma pequena diferença: um valor é atribuído à variável BRUXA. Qual valor? É isto que precisamos entender.

USR é uma função BASIC, assim precisa de um argumento (valor) e *devolve* um valor. Raciocinemos com a função SQR do BASIC. Em

LET BRUXA = SQR 256

256 é o argumento, e à variável BRUXA é atribuído o valor 16, pois $\sqrt{256} = 16$. Em

LET BRUXA = USRxxxxx

o argumento de USR é o endereço xxxx que fornecemos (16514, por exemplo), e este valor é carregado no par de registradores BC. O programa roda, e o valor atribuído à variável BRUXA é o valor contido no par de registradores BC *após* rodar o programa.

Vamos entender isto ainda melhor. Para tanto, vejamos um programa que não faz nada.

PROGRAMA 09

(1 byte)

16514 C9 RET

Para que você possa ver isto com facilidade no micro, *não vamos* usar o carregador ASSEMBLY.

Limpe a memória (NEW ou RAND USR 0), e digite a linha

1 REM TAN

Para obter TAN, você precisa do cursor **F**. NÃO digite T, depois A, depois N. TAN (função BASIC tangente) está na tecla E.

Explicações:

C9h = 201d = código do “caractere” TAN.

Digite agora:

10 LET L = USR 16514

20 PRINT L

Rode o programa, e você obterá no canto superior esquerdo do vídeo 16514!

Quando o micro encontra USR 16514, carrega o par de registradores BC com 16514. Assim sendo, temos B com 64 (40h) e C com 130 (82h), pois

$$16514 = 64 \times 256 + 130$$

Agora, o micro roda o programa ASSEMBLY do endereço 16514. Lá está RET. Assim, o programa retorna ao BASIC, sem que B e C sejam alterados. Logo, o valor do par BC ainda é 16514, e este valor é atribuído à variável L.

Assim, a linha 20 PRINT L imprime no canto superior esquerdo da tela o valor 16514.

Naturalmente poderia ter sido usado, em vez do programa BASIC escrito, o seguinte:

1 REM TAN

10 PRINT USR 16514

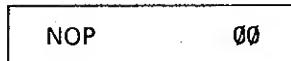
O efeito seria o mesmo.

5.3

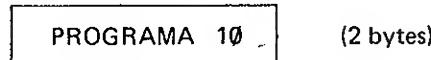
MAIS UMA INSTRUÇÃO ASSEMBLY - NOP

Vamos fazer outro programa, *que também não faz nada!* Para isto, precisamos de uma nova e sensacional instrução ASSEMBLY: a instrução NOP.

NOP é o mnemônico de NO OPERATION. Por pior que esteja o seu inglês, você percebeu: esta instrução não faz nada, e o programa simplesmente passa para a próxima instrução. Seu código é **00**.



Veja então o incrível



16514 00 NOP
16515 C9 RET

Novamente vamos colocar isto no micro SEM usar o carregador ASSEMBLY. Digite (após NEW ou RAND USR 0):

1 REM — TAN
→ sempre usaremos este símbolo para um espaço necessário

10 PRINT USR 16514

Verifique se você digitou corretamente a linha 1 REM espaço TAN. Rode o programa e... novamente 16514.

Claro, não? O programa é fantástico. A primeira instrução ordena que ele não faça nada, e a segunda, que retorne ao BASIC. Naturalmente os registros de B e C não são alterados; consequentemente o valor de retorno é 16514.

Naturalmente, podemos fazer um programa que altere o valor de retorno, por alterar o conteúdo dos registradores B e C. Por exemplo, veja o



(5 bytes)

16514 06 00 LD B,0
16516 0E 64 LD C,100
16518 C9 RET

Coloque este programa no micro com o carregador ASSEMBLY, e responda: a digitação de PRINT USR 16514, o que fornece? 100, é claro!

Como você sabe, ao encontrar USR 16514, o computador carrega BC com 16514, ou seja, B com 64 (40h) e C com 130 (82h), pois $64 \times 256 + 130 = 16514$.

Mas... a primeira instrução coloca 0 em B, e a segunda 100 em C.

Logo, o par BC agora contém 100, pois $0 \times 256 + 100 = 100$.

RET faz a volta ao BASIC, e o par BC agora contém 100. Assim, PRINT USR 16514 imprime na tela o valor atual do par de registradores BC: 100.

5.4

EXERCÍCIOS

- 25 Complete as lacunas após analisar este programa SEM RODÁ-LO. Justifique sua resposta.

Ø REM 3 caracteres quaisquer

16514	06	00	LD B,Ø
16516	C9		RET

PRINT USR 16514

O resultado impresso na tela é , pois $\times 256 +$ =

→ conteúdo decimal do registrador

→ conteúdo decimal do registrador

- 26 Faça o mesmo que no exercício anterior.

Ø REM 3 caracteres quaisquer

16514	0E	00
16516	C9	

PRINT USR 16514

O resultado impresso na tela é , pois $\times 256 +$ =

→ conteúdo decimal do registrador

→ conteúdo decimal do registrador

- 27 Ainda o mesmo exercício.

Ø REM 4 caracteres quaisquer

16514	01	22	00
16517	C9		

PRINT USR 16514

O resultado impresso na tela é , pois $\times 256 +$ =

→ conteúdo decimal do registrador

→ conteúdo decimal do registrador

- 28 Este programinha não faz nada de espetacular, é apenas um DESAFIO para vermos se você ainda lembra bem de JR, que estudamos no Capítulo 2. Desmonte-o, e diga SEM RODÁ-LO o que aparece na tela através de PRINT USR 16514.

Ø REM 9 caracteres quaisquer

16514	06	00
16516	0E	3A
16518	18	02

16520 ØE BA
16522 C9
PRINT USR 16514

5.5

FAZENDO LOAD DE UM REGISTRADOR PARA OUTRO

Em BASIC, você pode definir uma variável a partir de outra predefinida. Por exemplo:

10 LET BRUXA = 23

20 LET CALDO = BRUXA

Isto copia na variável CALDO o valor da variável BRUXA, que não precisa ser novamente explicitado na linha 20.

Em ASSEMBLY, isto é possível dentro de certos limites. É possível copiar o conteúdo de um registrador de 8 bits noutro registrador de 8 bits, ou seja, é válida a instrução

LD r,r

A instrução LD B,A copia no registrador B o conteúdo do registrador A. O conteúdo de B passa a ser igual ao conteúdo de A, que não é alterado.

Vamos supor que ANTES da instrução LD B,A, A contenha 23d e B contenha 128d.

APÓS a instrução LD B,A, tanto B quanto A conterão o valor 23d.

A tabela abaixo contém os códigos das instruções LD r,r.

LD	A	B	C	D	E	H	L
A	7F	78	79	7A	7B	7C	7D
B	47	40	41	42	43	44	45
C	4F	48	49	4A	4B	4C	4D
D	57	50	51	52	53	54	55
E	5F	58	59	5A	5B	5C	5D
H	67	60	61	62	63	64	65
L	6F	68	69	6A	6B	6C	6D

Observações:

- a coluna (vertical) da esquerda tem o "primeiro" registrador (destino).
- a linha (horizontal) superior tem o "segundo" registrador (fonte).

- c) se você entendeu, confira que o código hexa para LD B,H é 44, e para LD H,B é 60.
- d) observé que as instruções cujos códigos estão hachurados (LD A,A; LD B,B; LD C,C etc) NÃO fazem absolutamente NADA. São, de certa maneira, equivalentes a NOP.
- e) NÃO EXISTEM INSTRUÇÕES LD rr,rr NEM LD r,rr NEM LD rr,r. Sendo mais claro:
- LD BC,HL
LD A,HL } NÃO EXISTEM!
LD HL,A
- f) é possível *simular* LD BC,HL através de LD B,H e LD C,L.

5.6

EXERCÍCIOS

29 Desmonte o programa abaixo e dê a sua saída de vídeo SEM RODÁ-LO.

Ø REM 6 caracteres quaisquer

16514	3E	2A
16516	06	00
16518	4F	
16519	C9	

Após PRINT USR 16514, você obterá na tela, pois x 256 +
..... =

30 Mesmo exercício anterior.

Ø REM 7 caracteres quaisquer

16514	26	00
16516	1E	10
16518	44	
16519	4B	
16520	C9	

Após PRINT USR 16514, você obterá na tela, pois x 256 +
..... =

5.7

APRENDENDO A SOMAR

Vamos ver um último assunto antes de começarmos a aprender loops. Vamos aprender a somar em ASSEMBLY. Existem muitas instruções que somam, vamos aprender apenas a instrução do tipo

ADD A,r

Esta instrução adiciona ao acumulador A o conteúdo do registrador de 8 bits colocado à direita.

O registrador da esquerda é obrigatoriamente A, não existe uma instrução tipo ADD B,C.

O conteúdo do registrador da direita não é alterado.

Dois exemplos deixarão isto claro.

1º exemplo: Suponha que A contenha 50d, e C 22d. Após a instrução ADD A,C, A passa a ter 72d, e C continua tendo 22d. O Exercício 33 é baseado num programa com este esquema.

2º exemplo: Suponha que A contenha 200d, e que C contenha 100d. Após a instrução ADD A,C, A deveria conter 300d, o que é impossível, pois o maior número decimal que um registrador de 8 bits pode conter é 255. Assim, houve um OVERFLOW (transbordamento). No acumulador A "fica" o que excede 256, ou seja, $300 - 256 = 44$.

A bem da verdade, vamos dizer que o computador "anota" que houve transbordamento num registrador especial chamado F. Breve vamos estudar isto.

O Exercício 34 é baseado num programa com este esquema.

A tabela abaixo fornece os códigos hexa das instruções ADD A,r.

ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85

5.8

EXERCÍCIOS

31. Vejamos se você percebeu a analogia com BASIC. A instrução ADD A,C é, guardando as limitações que vimos, equivalente à instrução BASIC
32. Supondo que o conteúdo do registrador A seja menor ou igual a 127, qual o novo conteúdo de A após a instrução ADD A,A? Será sempre igual ao
33. Desmonte o programa abaixo, e diga qual a saída de vídeo, após PRINT USR 16514 SEM RODÁ-LO.

Ø REM caracteres quaisquer

16514	3E	22
16516	ØE	50
16518	81	
16519	4F	
16520	Ø6	ØØ
16522	C9	

Após PRINT USR 16514 você obterá na tela , pois x 256 + =

34 Mesmo exercício anterior, mas... CUIDADO COM O OVERFLOW!

Ø REM caracteres quaisquer

16514	3E	90
16516	06	A0
16518	80	
16519	4F	
16520	06	00
16522	C9	

Após a instrução do endereço 16518, o acumulador contém , pois = - 256.

Assim sendo, PRINT USR 16514 produz na tela , pois x 256 + =

35 Se, no programa anterior, esquecêssemos 16520 06 00, fazendo 16520 C9, qual seria a saída de vídeo? Seria , pois x 256 +

36 O programa abaixo simula a instrução ADD D,E, que NÃO existe. Desmonte-o e diga a saída de vídeo, SEM RODÁ-LO.

Ø REM caracteres quaisquer

16514	16	66
16516	1E	34
16518	7A	
16519	83	
16520	4F	
16521	06	00
16523	C9	

PRINT USR 16514

Após PRINT USR 16514, a saída é , pois x 256 + =

5.9

LOOPS – O CONCEITO DE CONTADOR

Analisemos o programa BASIC abaixo:

```
10 FOR B = 1 TO 32
20 PRINT "*";
30 NEXT B
```

O programa enche a primeira linha da tela com *. A variável de controle do loop (B) "conta" quantas vezes o * é impresso na tela. Isto é relativamente simples de obter em ASSEMBLY.

Um dos registradores do microprocessador Z-80 é privilegiado como contador — é o registrador B.

Assim, a instrução LD B,32, de uma certa forma, substitui a linha 10 do programa BASIC acima — ela faz de B um contador para 32 vezes.

Sabemos como substituir a linha 20. Senão, confira:

LD A,23
RST 10

Agora veremos como obter o loop — através da instrução DJNZ. Este impronunciável conjunto de letras é uma das poderosas instruções a nosso dispor.

DJNZ é o mnemônico de *Decrement B and Jump relative if Not Zero*. Tentando traduzir: decremente o contador B e faça um salto relativo se B não atingiu 0.

Se o contador B é inicializado com 14d, DJNZ vai decrementar para 13d, e saltar. Quando B atinge 1, e DJNZ é implementado, então B torna-se 0, e o programa segue para seu próximo endereço.

Importante: se B é inicializado com 0, então DJNZ “decrementa-o” para FFh, ou seja, 255, e o salto relativo acontece.

Como todo salto relativo, há a necessidade de 1 byte para fornecer a magnitude do salto.

Entre 00 e 7F temos um salto para frente, e entre 80 e FF temos um salto para trás, baseado na tabela de complemento de dois — tudo idêntico a JR, só que desta vez há um contador: B.

Vejamos o código hexa da instrução

DJNZ 10

Vamos completar então o nosso primeiro programa que faz loop, e um programa que, além da finalidade didática, tem finalidades práticas — depois as apresentaremos a você.

PROGRAMA 12 (8 bytes)

0 REM 8 caracteres quaisquer

16514	06	20	LD B,32
16516	3E	17	LD A,23
16518	D7		RST 10 ←
16519	10	FD	DJNZ -3
16521	C9		RET

Estudemos carinhosamente este programa.

- 1) LD B,32... faz de B o contador para 32 caracteres em uma linha.
- 2) LD A,23... carrega o acumulador com o código de *.
- 3) RST 10... coloca o asterisco na tela.
- 4) DJNZ -3... coloca o asterisco na tela mais 31 vezes, pois volta ao endereço de RST 10.
- 5) RET... após o contador B ser zerado, volta ao BASIC.

Estude cuidadosamente a estrutura deste programa — ele é importantíssimo. Quando tiver certeza de tê-lo entendido, passe aos exercícios.

Lembra-se de que dissemos que este programa tinha finalidades práticas? E tem! Eu uso freqüentemente esta sub-rotina em meus programas BASIC para produzir uma linha de separação, muito útil quando entramos com diversos dados. Para chamar a sub-rotina, basta introduzir o RAND USR 16514 como *linha de programa*, e lá está a linha de separação *instantaneamente* na tela. O caractere Ø3 é útil para construir a linha.

5.10

EXERCÍCIOS

- 37 Supondo que o programa 12 esteja no seu computador, digite (modo direto) PRINT USR 16514 (ao invés de RAND USR 16514). Você sabe explicar o número 13Ø que aparece na tela após a linha de *? Outra maneira de obter este número 13Ø seria fazer (sempre modo direto) LET L = USR 16514 (que coloca a linha na tela) seguido de PRINT L (que “apaga” a linha e imprime 13Ø). Insistimos: você sabe explicar este número 13Ø?
- 38 Complete a rotina abaixo para que ela produza as 5 linhas superiores da tela cheias com “espaço inverso”.

Ø REM caracteres quaisquer

16514	LD B,.....
165...	LD A,.....
165...	RST 1Ø
165...	DJNZ.....
165...	RET

Que número (decimal) está no endereço 16515? Justifique-o.

- 39 Esta mesma rotina pode produzir as 8 linhas superiores da tela cheias com um caractere a nossa escolha, por exemplo + .

Há que usar-se um truque, pois $8 \times 32 = 256$, e você NÃO pode fazer LD B,256.

Releia a teoria de DJNZ, e descubra o valor surpreendente com que se deve carregar B...

Ø REM caracteres quaisquer

16514	LD B,.....
165...	LD A,.....
165...	RST 1Ø
165...	DJNZ,.....
165...	RET

- 40 Bem, como vimos no exercício anterior, o maior número de linhas que pode ser preenchido com um caractere a nossa escolha usando *uma* instrução RST 1Ø é 8. Como fazer então para produzir 9 linhas preenchidas?

A rotina abaixo resolve o problema, usando *duas* instruções RST 10. Calcule cuidadosamente o número com que deve ser carregado o contador B para que tenhamos 9 linhas preenchidas com **>**. *Cuidado com o salto relativo.*

Ø REM caracteres quaisquer

16514	LD B,.....
165...	LD A,.....
165...	RST 10
165...	RST 10
165...	DJNZ,.....
165...	RET

Explique detalhadamente o número com que você carregou o contador B.

- 41 Raciocine em função do Exercício 39, e responda: qual o maior número de linhas que pode ser preenchido usando apenas *duas* instruções RST 10?

Complete a rotina abaixo para que produza este efeito (o maior número possível de linhas usando duas instruções RST 10) com o caractere **█**.

Ø REM caracteres quaisquer

16514	LD B,.....
165...	LD A,.....
165...	RST 10
165...	RST 10
165...	DJNZ,.....
165...	RET

Mostre o cálculo que evidencia o número de linhas que serão preenchidas.

- 42 Finalmente, vamos preparar um programa que enche a tela: são necessárias *quatro* instruções RST 10. Não é possível com três, pois 704 – o número de caracteres da tela toda – não é múltiplo de três.

A rotina que você vai escrever tem importância teórica: é a mais curta para a produção deste efeito!

Complete-a para que encha a tela com o caractere 8. Aprecie a *velocidade* e a *economia de memória!*

Ø REM caracteres quaisquer

16514	LD B,.....
165...	LD A,.....
165...	RST 10
165...	RST 10
165...	RST 10
165...	RST 10
165...	DJNZ,.....
165...	RET

Mostre o cálculo do número com que você carregou o contador.

43 Nem só de efeitos de tela é a nossa vida... Vamos agora ver um programa "matemático".

O livro "Programação em Assembler e Linguagem de Máquina", de David C. Alexander, editado em 1984 pela CAMPUS, é voltado para a linha TRS-80 (CP-300, CP-500, DGT-100 etc.). Todos estes micros "grandes" usam o mesmo microprocessador Z-80 usado pelos nossos "pequenos" SINCLAIR. Logo, os programas podem ser adaptados, desde que conheçamos os endereços da ROM...

No Capítulo 15, página 104, encontramos o seguinte exemplo, através do qual o autor explica a instrução DJNZ:

	LD A,00h	; coloca zero em A
	LD H,05h	; armazena o valor 5 no registrador H
	LD B,03h	; carrega o número de voltas do loop
LOOP	ADD A,H	; A agora contém mais 5
	DJNZ LOOP	; se B não for zero, desvie para a soma indicada pelo rótulo LOOP
	JP NEXT	; vá para a próxima rotina

Vamos transformar este exemplo em algo que rode em nosso micro. Eu disse vamos?

Desculpe: VOCÊ VAI!

Ø REM caracteres quaisquer

16514	LD A,0
16516	LD H,5
16518	LD B,3
LOOP	16520	ADD A,H ←
	16521	DJNZ.... ↗
	16523	LD C,A
	16524	RET

PRINT USR 16514

Responda ANTES de rodar o programa:

- Qual a finalidade de LD C,A?
- Por que não é necessário fazer-se LD B,0 para preparar a saída?
- Qual a saída do programa, ou seja, que número vai impresso através de PRINT USR 16514?

44 O último exercício do capítulo é uma homenagem a LEWIS CARROLL. Sim, este mesmo, o autor de *Alice no País das Maravilhas!* Se você não sabia, seu nome não era nem LEWIS nem CARROLL: era Charles Lutwidge Dodgson, e era professor de matemática em Oxford e autor de várias obras científicas. Mas notabilizou-se por suas histórias infantis, que estão, para o observador atento, repletas de matemática e de lógica matemática.

De um de seus livros, *Através do Espelho (Through the Looking-Glass, de 1871)*, pinçamos esta curiosa passagem:

- Quanto é um mais um?
- Eu não sei — disse Alice. — Perdi a conta.
- Ela não sabe somar — disse a Rainha Vermelha.

Bem, vamos ajudar Alice! Prepare um programa que faça a soma proposta ($1 + 1 + 1 \dots$) e dê a resposta através de PRINT USR 16514.

RESPOSTAS DOS EXERCÍCIOS

- 25 O resultado impresso na tela é 130, pois

$$0 \times 256 + 130 = 130$$

↓
→ conteúdo decimal do registrador B
→ conteúdo decimal do registrador C

O conteúdo do registrador B foi alterado pela primeira instrução do programa, mas o de C não! Assim, ainda contém 130.

- 26 16514 0E 00 LD C,0
16516 C9 RET

O resultado impresso na tela é 16384, pois

$$64 \times 256 + 0 = 16384.$$

↓
→ conteúdo decimal do registrador B
→ conteúdo decimal do registrador C

O conteúdo do registrador C foi alterado pela primeira instrução do programa, mas o de B não! Assim, ainda contém 64.

- 27 0 REM 4 caracteres quaisquer

16514 01 22 00 LD BC,34
16517 C9 RET

O resultado impresso na tela é 34, pois

$$0 \times 256 + 34 = 34$$

↓
→ conteúdo decimal do registrador B
→ conteúdo decimal do registrador C

- 28 16514 06 00 LD B,0
16516 0E 3A LD C,58
16518 18 02 JR 2
16520 0E BA LD C,186
16522 C9 RET

A primeira instrução "zera" o conteúdo de B.

A segunda carrega C com 58.

A terceira salta os próximos 2 endereços! Logo, a quarta instrução (LD C,186) NÃO se realiza, e o programa retorna (RET) ao BASIC, com B contendo 0 e C contendo 58 (e não 186).

Assim sendo, PRINT USR 16514 produz 58, pois

$$0 \times 256 + 58 = 58.$$

29	16514	3E	2A	LD A,42
	16516	06	00	LD B,0
	16518	4F		LD C,A
	16519	C9		RET

Como você pode ver, B contém 0, e C contém 42 (assim como A). Logo, PRINT USR 16514 produz 42, pois

$$0 \times 256 + 42 = 42$$

30	16514	26	00	LD H,0
	16516	1E	10	LD E,16
	16518	44		LD B,H
	16519	4B		LD C,E
	16520	C9		RET

Como você pode ver, B contém 0 (assim como H) e C contém 16 (assim como E). Logo, PRINT USR 16514 produz 16, pois

$$0 \times 256 + 16 = 16$$

31 LET A = A + C

32 Dobro do conteúdo anterior de A.

33 Ø REM 9 caracteres quaisquer

16514	3E	22	LD A,34
16516	0E	50	LD C,80
16518	81		ADD A,C
16519	4F		LD C,A
16520	06	00	LD B,0
16522	C9		RET

Após PRINT USR 16514, você obterá na tela 114, pois

$$0 \times 256 + 114 = 114$$

34 Ø REM 9 caracteres quaisquer

16514	3E	90	LD A,144
16516	06	A0	LD B,160
16518	80		ADD A,B
16519	4F		LD C,A
16520	06	00	LD B,0
16522	C9		RET

Após a instrução do endereço 16518, o acumulador contém 48, pois

$$48 = 144 + 160 - 256.$$

Assim sendo, PRINT USR 16514 produz na tela 48, pois

$$0 \times 256 + 48 = 48.$$

35 Bem, B conteria 160 e C conteria 48, como explicado acima. Logo, a saída após PRINT USR 16514 seria 41008, pois $160 \times 256 + 48 = 41008$.

36 Ø REM 10 caracteres quaisquer

16514	16	66	LD D,102
16516	1E	34	LD E,52
16518	7A		LD A,D
16519	83		ADD A,E
16520	4F		LD C,A
16521	06	00	LD B,Ø
16523	C9		RET

Após PRINT USR 16514, a saída é 154, pois

$$\text{Ø} \times 256 + 154 = 154.$$

37 Lembre-se de que USR é uma função BASIC. Seu argumento é o endereço inicial, que é carregado no par de registradores BC. O resultado (saída) é o valor contido em BC *após* a execução da rotina. Assim, ao iniciar a nossa rotina, BC é carregado com 16514, logo B e C contêm respectivamente 64 e 130, pois

$$64 \times 256 + 130 = 16514.$$

Ao rodar o programa, a instrução DJNZ zera o contador B, mas C não é alterado pela rotina! Logo, continua contendo 130!

Assim, o valor do par BC ao final do programa é

$$\text{Ø} \times 256 + 130 = 130$$

Como este valor não nos interessa para nada, digitaremos RAND USR 16514 para que ele *não* seja impresso.

38 Ø REM 8 caracteres quaisquer

16514	06	A0	LD B,160
16516	3E	80	LD A,128
16518	D7		RST 10 ←
16519	10	FD	DJNZ -3 ←
16521	C9		RET

Em 16515 está 160, que é igual a 5 × 32 (5 linhas × 32 caracteres por linha).

39 Ø REM 8 caracteres quaisquer

16514	06	00	LD B,Ø
16516	3E	95	LD A,149
16518	D7		RST 10 ←
16519	10	FD	DJNZ -3 ←
16521	C9		RET

Explicações: se B está carregado com Ø e é decrementado, “ficaria” com -1. Mas, pela tabela de complemento de dois (Apêndice 4), -1 = FF = 255. E assim continuará sendo decrementado de 255 até Ø novamente, totalizando 256.

Resumindo: o maior número de vezes que uma instrução, ou conjunto de instruções, pode ser repetida através de DJNZ é 256. Um truque interessante...

Assim, é fácil ver agora que o maior número de linhas que podemos preencher com *apenas uma* instrução RST 10 é 8, pois

$$256 \div 32 = 8 \text{ (256 impressões de caractere} \div 32 \text{ caracteres por linha} = 8 \text{ linhas}).$$

40 Ø REM 9 caracteres quaisquer

16514	06	90	LD B,144
16516	3E	9B	LD A,155
16518	D7		RST 10 ←
16519	D7		RST 10
16520	10	FC	DJNZ -4
16522	C9		RET

O contador B deve ser carregado com $9 \times 32/2 = 144$, pois 9 linhas x 32 caracteres por linha/2 impressões (RST 10) por volta do loop = 144.

41 Ø REM 9 caracteres quaisquer

16514	06	00	LD B,0
16516	3E	8E	LD A,142
16518	D7		RST 10 ←
16519	D7		RST 10
16520	10	FC	DJNZ -4
16522	C9		RET

Isto produz 16 linhas preenchidas pois $256 \times 2/32 = 16$. Explicando: 256 repetições de instruções (LD B,0 seguido de DJNZ) x 2 impressões (RST 10) por volta do loop $\div 32$ caracteres por linha = 16.

42 Ø REM 11 caracteres quaisquer

16514	06	B0	LD B,176
16516	3E	08	LD A,8
16518	D7		RST 10 ←
16519	D7		RST 10
16520	D7		RST 10
16521	D7		RST 10
16522	10	FA	DJNZ -6
16524	C9		RET

Cálculo do número com que carregamos o contador:

$$22 \times 32 \div 4 = 176$$

pois são 22 linhas x 32 caracteres por linha $\div 4$ impressões (RST 10) por volta do loop.

43 Ø REM 11 caracteres quaisquer

16514	3E	00	LD A,0
16516	26	05	LD H,5
16518	06	03	LD B,3
16520	84		ADD A,H ←
16521	10	FD	DJNZ -3
16523	4F		LD C,A
16524	C9		RET

- LD C,A prepara a saída, colocando em C o valor presente no acumulador A.
- Não é necessário fazer LD B,0, pois a instrução DJNZ já fez isto.
- 15.

44 Ø REM 11 caracteres quaisquer

16514	3E	00	LD A,Ø ; inicializa o acumulador
16516	26	01	LD H,1 ; coloca em H a parcela
16518	06	0A	LD B,10 ; faz de B o contador do loop
LOOP	16520	84	ADD A,H ; soma
	16521	10	DJNZ -3 ; volta para somar de novo
	16523	4F	LD C,A ; coloca A em C para fazer a saída
	16524	C9	RET ; volta ao BASIC

PRINT USR 16514

6.1

INTRODUÇÃO

A instrução DJNZ usa o registrador B como contador. Entretanto, às vezes isto é impossível, pois B pode estar sendo usado para outra coisa. Certas vezes pode ser preferível usar outro registrador de 8 bits como contador, em vez de B.

Para isto, é novamente necessário que aumentemos o-conjunto de instruções ASSEMBLY conhecidas por nós. É o que faremos nos próximos itens.

6.2

INSTRUÇÕES INC r E DEC r

Em um loop do tipo

FOR A = 1 TO 10

.....

NEXT A

a variável de controle vai sendo incrementada, desde 1 até 10, passando por 2, 3, 4,..., 9.

Se fizermos

FOR A = 10 TO 1 STEP -1

.....

NEXT A

a variável de controle vai sendo decrementada, desde 10 até 1, passando por 9, 8, 7,..., 2.

A instrução DJNZ decremente automaticamente o contador B. Se desejamos fazer isto com outro registrador de 8 bits, precisamos fazê-lo nós mesmos.

A instrução tem a forma INC r ou DEC r, e tem a seguinte correspondência BASIC:

INC A..... LET A = A + 1

DEC A..... LET A = A - 1

Se L estava carregado com 04, após INC L tem 05. Se L continha FF, após INC L contém 00.

Se L estava carregado com 04, após DEC L tem 03. Se L continha 00, após DEC L contém FF.

Observe os códigos hexa das instruções:

INC A	3C	DEC A	3D
INC B	04	DEC B	05
INC C	0C	DEC C	0D
INC D	14	DEC D	15
INC E	1C	DEC E	1D
INC H	24	DEC H	25
INC L	2C	DEC L	2D

Comprove as nossas afirmações da página anterior rodando os seguintes programas.

PROGRAMA 13 (5 bytes)

16514	01	FF	00	LD BC,255
16517	0C			INC C
16518	C9			RET

PRINT USR 16514

Observe um truque. Poderíamos ter feito LD B,0 e LD C,255, o que seria mais "compreensível", mas gastaria um byte a mais.

PROGRAMA 14 (5 bytes)

16514	01	00	00	LD BC,0
16517	0D			DEC C
16519	C9			RET

PRINT USR 16514

Naturalmente os resultados são 0 e 255, respectivamente.

Bem, chegamos a um instante importante. Está na hora de você ler cuidadosamente o Apêndice 1, detendo-se *especialmente* nas explicações sobre o registrador de estado F.

NÃO PROSSIGA NA LEITURA DESTE CAPÍTULO SEM ANTES LER O APÊNDICE 1!

Pois bem, agora que você conhece o registrador F, saiba que INC r e DEC r afetam a ZERO FLAG. (Para um estudo detalhado de quais instruções afetam mais flags, vide Apêndice 11.)

Logo, podemos detectar quando o registrador-contador atinge 0 e um loop. Isto nos permite fazer SALTOS CONDICIONADOS. Este é o nosso próximo assunto.

6.3

SALTOS ABSOLUTOS CONDICIONADOS

Com base nas flags afetadas por cada instrução, são possíveis os seguintes saltos condicionados (após o código da instrução, há necessidade de 2 bytes para o endereço).

- JP P,NN (Jump if Plus) — salta, se o *último* resultado é positivo.
- JP M,NN (Jump if Minus) — salta, se o *último* resultado é negativo.
- JP Z,NN (Jump if Zero) — salta, se o *último* resultado é zero.
- JP NZ,NN (Jump if Not Zero) — salta, se o *último* resultado não é zero.
- JP PO,NN (Jump if Parity is Odd) — salta, se após a última instrução a paridade é ímpar (odd) ou se *não* houve transbordamento (overflow).
- JP PE,NN (Jump if Parity is Even) — salta, se após a última instrução a paridade é par (even) ou se *houve* transbordamento (overflow).
- JP C,NN (Jump if Carry) — salta, se no último resultado houve transporte (carry).
- JP NC,NN (Jump if No Carry) — salta, se no último resultado não houve transporte (carry).

A tabela abaixo fornece os códigos hexa destas instruções.

JP P	F2
JP M	FA
JP Z	CA
JP NZ	C2
JP PO	E2
JP PE	EA
JP C	DA
JP NC	D2

6.4

SALTOS RELATIVOS CONDICIONADOS

Embora não tão variadamente como nos saltos absolutos condicionados, saltos relativos condicionados também são possíveis. E são utilíssimos...

Após o código da instrução, usaremos mais um byte para dar o deslocamento do salto, para frente ou para trás.

A tabela abaixo fornecerá os códigos hexa das instruções permitidas.

JR Z	28
JR NZ	20
JR C	38
JR NC	30

6.5

A INSTRUÇÃO COMPARE (CP)

A instrução COMPARE (CP) calcula o resultado de "acumulador menos registrador ou dado", e altera as flags convenientemente. O resultado da comparação (subtração) não é guardado em lugar nenhum. Vejamos os códigos hexa das instruções que nos interessam.

CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP N	FE (+1 byte)

6.6

USO DE CONTADORES

Compare os esquemas abaixo.

LD B,.....
rotina ←
DJNZ []

LD L,.....
rotina ←
DEC L
JR NZ,....

Em termos de funcionamento, estas duas possibilidades são iguais. DJNZ é apenas mais econômico.

Outra possibilidade:

LD C,0
rotina ←
INC C
CP....
JR NZ,....

Também é equivalente. Mais trabalhosa, mas... necessária às vezes.

Vamos fazer alguns problemas como exemplo. Como primeira idéia, vamos fazer um loop dentro de outro.

Estabeleçamos uma rotina que encha uma linha com ADAD...

LD B,16 ; haverá 16 "pares" AD
CARGA LD A,"A" ; coloca no acumulador o caractere A
RST 10 ; imprime A na tela
LD A,"D" ; coloca no acumulador o caractere D
RST 10 ; imprime D na tela
DJNZ CARGA

Agora, que temos uma linha, precisamos de um contador para 22 linhas. Seja C este contador. O esquema será:

LINHA LD C,22
CARGA LD B,16
LD A,"A"
.

DJNZ CARGA
DEC C
JR NZ LINHA
RET

E assim fizemos o

PROGRAMA 15

(16 bytes)

LINHA	16514	0E	16	LD C,22
CARGA	16516	06	10	LD B,16 ←
	16518	3E	26	LD A,38 ←
	16520	D7		RST 10
	16521	3E	29	LD A,41
	16523	D7		RST 10
	16524	10	F8	DJNZ -8
	16526	0D		DEC C
	16527	20	F3	JR NZ, -13
	16529	C9		RET

Rode o programa, modifique os caracteres se quiser, mas estude detalhadamente o que fizemos. Os exercícios serão verdadeiros desafios...

6.7

EXERCÍCIOS

45 Faça um programa que encha a tela da seguinte maneira:

- 1 linha "espaço invertido"
- 1 linha "caractere 8"
- 1 linha "espaço invertido"
- 1 linha "caractere 8"
-

até o fim da tela.

SUGESTÃO DE ESTRUTURA DO PROGRAMA

- 1) Contador C para 16 conjuntos de pares de linhas
- 2) Loop para a linha preta com contador B e DJNZ
- 3) Loop para a linha cinza com contador B e DJNZ
- 4) DEC C
- 5) JR NZ voltando para o item 2
- 6) RET (não esqueça! ... ou CRASH!)

Se você acertou, o programa leva 20 bytes. Insista. É muito importante que você aprenda a trabalhar com loops. Não "cole" a resposta. Você só aprende o que faz.

- 46 No Exercício 15 do Capítulo 3, surgiu a idéia de escrever o alfabeto na tela, idéia esta que foi abandonada, pois levaria 79 bytes (!) com a técnica de que dispunhamos até então.

No Exercício 18 do Capítulo 3, efetivamente colocamos o alfabeto na tela, levando 36 bytes.

Com o que sabemos agora, levaremos apenas 9 bytes!

Nossas sugestões para o seu programa:

- o código do caractere A é 38 (26h); carregue o acumulador com ele.
- com RST 10 coloque-o na tela.
- o código do caractere Z é 63 (3F)
- assim, *incremente* o acumulador A, compare com 64 (40h) (POR QUÊ?), e dê um JR NZ para a instrução RST 10, que colocará B, C, ... etc. na tela, até Z.
- não esqueça do RET.

Escreva seu programa, confira-o (9 bytes) e rode-o. A velocidade do loop é impressionante. Se desejar, compare-o com o mesmo programa escrito em BASIC:

```
10 FOR A = 38 TO 63  
20 PRINT CHR$ A;  
30 NEXT A
```

Além da velocidade, a economia de memória também é impressionante — este programa BASIC "gasta" 41 bytes (!).

- 47 Aproveitando a idéia do Exercício 46, você deve escrever mais alguns programas. Primeiramente escreva o alfabeto centrado na tela. Relembre PRINT AT em linguagem de máquina, no Capítulo 4.
- 48 Escreva o alfabeto de trás para frente, ou seja, começando de Z e terminando em A.
- 49 Escreva o alfabeto, de A até Z, mas em vídeo inverso.
- 50 Escreva o alfabeto, de trás para frente, centrado na tela, em vídeo inverso.
- 51 Os micros de linha Sinclair apresentam um total de 128 caracteres imprimíveis, que não seguem a convenção ASCII. Seus códigos vão de 0 a 63, para os em vídeo direto, e de 128 a 191, para os em vídeo inverso. Como você pode perceber, a diferença numérica entre os códigos de um caractere e seu inverso é constante e igual a 128 (80h). Para melhor entendimento, consulte o Apêndice 2.

Escreva um programa que escreva na tela os 128 caracteres imprimíveis (você vai ocupar exatamente 4 linhas). Sugestão (incompleta):

```
LD A,0
RST 10
INC A
CP 64
JR NZ
ADD A,... (você precisa passar de 64 para 128)-
RST 10
etc.
```

Bom trabalho...

- 52 Escreva um programa que encha a tela com o seguinte esquema:

```
1ª linha → 32 letras A
2ª linha → 32 letras B
3ª linha → 32 letras C
etc ...
```

Este é sem sugestão. Divirta-se!

- 53 O programa que o convidamos a escrever agora *não tem* uma saída de vídeo bonita — é no máximo curiosa. Mas... tem uma novidade para você — a idéia de preservar o valor do contador noutro registrador fora do loop, para este valor poder ser incrementado ou decrementado depois. Usaremos esta mesma idéia mais tarde num programa que tem uma linda saída de vídeo.

A idéia é a seguinte (consulte o Apêndice 2 para entendê-la claramente): deverão ser impressos na tela *um* ponto, *dois* números 0, *três* números 1 etc. até *trinta e sete* letras Z.

O número de caracteres a serem impressos é $\frac{(1 + 37) \times 37}{2} = 19 \times 37 = 703$, o que *cabe* na tela (por um!).

Eis a estrutura do programa que você deverá escrever, com alguns comentários. Deixamos todos os cálculos para você.

INÍCIO	→	LD A,código do caractere "ponto" ; inicializa o acumulador.
		LD C,1 ; ponto será impresso uma vez.
LOOP	→	LD B,C ; coloca o contador em B.
IMPRIME	→	RST 10 ; imprime.
		DJNZ IMPRIME ; nº de vezes da impressão
		INC C ; incrementa o contador.
		INC A ; incrementa o acumulador.
		CP..... ; verifica se "passou" de Z.
		JR Z FIM ; se "passou", volta ao BASIC.
		JR LOOP ; volta ao loop.
FIM	→	RET ; volta ao BASIC.

Em tempo: a fórmula que usamos para achar o número de caracteres impressos nada mais é que a fórmula da soma dos termos de uma P. A. ...

RESPOSTAS DOS EXERCÍCIOS

45 Ø REM 20 caracteres quaisquer

16514	ØE	ØB	LD C,11	
16516	Ø6	2Ø	LD B,32	←
16518	3E	8Ø	LD A,128	
16520	D7		RST 1Ø	←
16521	1Ø	FD	DJNZ -3	
16523	Ø6	2Ø	LD B,32	
16525	3E	Ø8	LD A,8	
16527	D7		RST 1Ø	←
16528	1Ø	FD	DJNZ -3	
16530	ØD		DEC C	
16531	2Ø	EF	JR NZ,-17	
16533	C9		RET	

46 Ø REM 9 caracteres quaisquer

16514	3E	26	LD A,38	
16516	D7		RST 1Ø	←
16517	3C		INC A	
16518	FE	4Ø	CP 64	
16520	2Ø	FA	JR NZ,-6	
16522	C9		RET	

Por que CP 64? Como sabemos, a instrução CP afeta a ZERO FLAG (ver Apêndice 11). Assim, após uma instrução CP, podemos fazer JR NZ, JR Z, JP NZ ou JP Z.

Quando o acumulador atingir o valor 64, a instrução CP provocará o resultado Ø (lembre-se de que CP é uma subtração). A instrução JR NZ,-6 não será implementada, e o programa retornará ao BASIC.

47 Como o alfabeto tem 26 letras, vamos escrevê-lo a partir de 11,3; ou seja, o comando BASIC equivalente seria

PRINT AT 11,3;"ABC Z"

Assim sendo, B = 11 e C = 3 para chamarmos a rotina 2293, que posiciona o cursor de impressão. Você reviu o Capítulo 4?

Ø REM 15 caracteres quaisquer

16514	Ø1	Ø3	ØB	LD BC,xxxx (B = 11 e C = 3)
16517	CD	F5	Ø8	CALL 2293
16520	3E	26		LD A,38
16522	D7			RST 1Ø
16523	3C			INC A
16524	FE	4Ø		CP 64
16526	2Ø	FA		JR NZ,-6
16528	C9			RET

48 Ø REM 9 caracteres quaisquer

16514	3E	3F	LD A,63
16516	D7		RST 1Ø
16517	3D		DEC A
16518	FE	25	CP 37
16520	2Ø	FA	JR NZ,-6
16522	C9		RET

Certifique-se de que entendeu qual o motivo de LD A,63 e CP 37.

49 Ø REM 9 caracteres quaisquer

16514	3E	A6	LD A,166
16516	D7		RST 1Ø
16517	3C		INC A
16518	FE	CØ	CP 192
16520	2Ø	FA	JR NZ,-6
16522	C9		RET

Certifique-se de que entendeu o motivo de LD A,166 e CP 192. Consulte o Apêndice 2 em caso de dúvida!

50 Ø REM 15 caracteres quaisquer

16514	Ø1	Ø3	ØB	LD BC,xxxx (B = 11 e C = 3)
16517	CD	F5	Ø8	CALL 2293
16520	3E	BF		LD A,191
16522	D7			RST 1Ø
16523	3D			DEC A
16524	FE	A5		CP 165
16526	2Ø	FA		JR NZ,-6
16528	C9			RET

51 Ø REM 16 caracteres quaisquer

16514	3E	ØØ	LD A,Ø
16516	D7		RST 1Ø
16517	3C		INC A
16518	FE	4Ø	CP 64
16520	2Ø	FA	JR NZ,-6
16522	87		ADD A,A
16523	D7		RST 1Ø
16524	3C		INC A
16525	FE	CØ	CP 192
16527	2Ø	FA	JR NZ,-6
16529	C9		RET

Também seria possível, em vez de ADD A,A (87h), ADD A,64 (C6 4Ø). Mas como fizemos é mais curto (1 byte apenas) e mais elegante.

52 Como não fizemos nenhuma sugestão, este programa vai comentado.

Ø REM 14 caracteres quaisquer

16514	0E	16	LD C,22	; contador para 22 linhas
16516	3E	26	LD A,38	; inicializa A com o código de "A"
16518	06	20	LD B,32	; contador de caracteres na linha
16520	D7		RST 10	; imprime a letra ↪
16521	10	FD	DJNZ -3	; completa a linha ↩
16523	3C		INC A	; próxima letra
16524	0D		DEC C	; decrementa contador de linhas
16525	20	F7	JR NZ,-9	; próxima linha
16527	C9		RET	; volta ao BÁSIC

53 Ø REM 17 caracteres quaisquer

INÍCIO	→	16514	3E	1B	LD A,27
		16516	0E	01	LD C,1
LOOP	→	16518	41		LD B,C
IMPRIME	→	16519	D7		RST 10
		16520	10	FD	DJNZ -3
		16522	0C		INC C
		16523	3C		INC A
		16524	FE	40	CP 64
		16526	28	02	JR Z,2
		16528	18	F4	JR,-12
FIM	→	16530	C9		RET

7.1

INTRODUÇÃO

Neste capítulo vamos estudar a tela de TV, a saída de vídeo fornecida pelos micros de lógica SINCLAIR.

Trabalhar diretamente com a memória de vídeo nos dará uma incrível versatilidade, naturalmente com o ônus de aprendermos novas instruções ASSEMBLY e melhorarmos nosso entendimento sobre as variáveis do sistema.

Normalmente consideramos a tela como tendo 24 linhas de 32 colunas cada uma, sendo as duas linhas inferiores reservadas para edição — são as linhas usadas pelo cursor.

O número de linhas usadas pelo cursor é controlado pela variável do sistema DF.SZ, cujo endereço é 16418 (4022h). Ela normalmente contém o valor 2. Ou seja, se você ligar o computador e digitar

PRINT PEEK 16418

obterá na tela o valor 2.

Como podemos fazer PEEK em linguagem de máquina? Observe o esquema abaixo, entendendo por NN o endereço onde se quer fazer PEEK.

INSTRUÇÃO	CÓDIGO HEXA	CORRESPONDÊNCIA BASIC
LD A,(NN)	3A	LET A=PEEK NN
LD BC,(NN)	ED 4B	LET C=PEEK NN LET B=PEEK (NN+1)
LD DE,(NN)	ED 5B	LET E=PEEK NN LET D=PEEK (NN+1)
LD HL,(NN)	2A	LET L=PEEK NN LET H=PEEK (NN+1)

Preste muita atenção aos parênteses! LD A,16418 exprime algo impossível: carregar A com o valor 16418 (> 255); enquanto que LD A,(16418) é perfeitamente possível: carrega A com o conteúdo (PEEK) do endereço 16418.

Vamos fazer PRINT PEEK 16418 em linguagem de máquina. Várias soluções são possíveis. Vejamos algumas.

- a) LD A,(16418) ATENÇÃO PARA A INVERSÃO DO ENDEREÇO!
LD C,A
LD B,0

PROGRAMA 16 – a

(7 bytes)

Ø REM 7 caracteres quaisquer

16514	3A	22	40	LD A,(16418)
16517	4F			LD C,A
16518	06	00		LD B,Ø
16519	C9			RET

PRINT USR 16514 produz o valor 2.

- b) LD BC,(16418)
LD B,Ø

Ao fazermos LD BC,(16418), fizemos LET C=PEEK 16418 e LET B=PEEK 16419. Este valor de B não nos interessa, logo deve ser "zerado".

PROGRAMA 16 – b

(7 bytes)

16514	ED	4B	22	40	LD BC,(16418)
16518	06	00			LD B,Ø
16519	C9				RET

- c) LD HL,(16418)
LD C,L
LD B,Ø

PROGRAMA 16 – c

(7 bytes)

16514	2A	22	40	LD HL,(16418)
16517	4D			LD C,L
16518	06	00		LD B,Ø
16519	C9			RET

Naturalmente os três programas são equivalentes. Fizemos questão de apresentar os três para você vá ganhando flexibilidade. Voltemos ao nosso assunto.

7.2

ALTERANDO AS LINHAS DE EDIÇÃO

Podemos alterar o número de linhas de edição, inclusive suprimindo-as, fazendo um POKE em 16418. Através de POKE 16418,Ø anulamos as duas linhas de edição, passando a ter acesso às 24 linhas da tela.

No entanto, você NÃO PODE fazer um INPUT após POKE 16418,Ø, senão... CRASH!

Experimente este programa BASIC.

```
10 POKE 16418,0
20 FOR L=0 TO 23
30 PRINT AT L,L;"**"
40 NEXT L
```

Ele imprime 24 * em diagonal na tela, parando com código 0/40. No entanto, se você acrescentar

```
50 INPUT I$
60 CLS
```

e rodar o programa, normalmente perderá acesso ao teclado, permanecendo a tela aparentemente normal por alguns segundos antes do CRASH.

Como podemos fazer POKE em linguagem de máquina? Observe o esquema abaixo, entendendo por NN o endereço onde se quer fazer POKE.

INSTRUÇÃO	CÓDIGO HEXA	CORRESPONDÊNCIA BASIC
LD (NN),A	32	POKE NN,A
LD (NN),BC	ED 43	POKE NN,C
LD (NN),DE	ED 53	POKE NN+1,B
LD (NN),HL	22	POKE NN,E
		POKE NN+1,D
		POKE NN,L
		POKE NN+1,H

Assim, podemos substituir a linha 10 do programa BASIC do item anterior por um programa ASSEMBLY que faça POKE 16418,0:

```
LD A,0
LD (16418),A
```

PROGRAMA 17

(6 bytes)

0 REM 6 caracteres quaisquer

16514	3E	00		LD A,0
16516	32	22	40	LD (16418),A
16519	C9			RET

Após dar entrada a este programa, que agora "mora" na linha 0, digite o nosso velho programa BASIC, com uma nova linha 10.

```
10 RAND USR 16514
20 FOR L=0 TO 23
30 PRINT AT L,L;"**"
40 NEXT L
```

e veja que *funciona!*

7.3

A MEMÓRIA DA TELA

Podemos encarar a tela como sendo um retângulo de 32 (colunas) x 24 (linhas) posições. No entanto, o computador arquiva a tela de maneira completamente diferente. Vamos começar a entender este importantíssimo tópico.

O arquivo de imagem é seqüencial, iniciando-se por um caractere 118 (NEW LINE), seguido de 32 bytes para a 1^a linha, um caractere 118 para indicar o fim da 1^a linha, 32 bytes para a 2^a linha, um caractere 118 para indicar o fim da 2^a linha, 32 bytes para a 3^a linha, e assim sucessivamente até a 24^a linha (as duas linhas de edição pertencem ao arquivo de imagem).

O endereço inicial deste arquivo é variável, sendo dado através da variável do sistema D.FILE, uma variável de 2 bytes, cujo endereço inicial é 16396 (400C).

Logo, o endereço inicial do arquivo de imagem pode ser obtido em BASIC através de

PRINT PEEK 16396+256*PEEK 16397

ou, em linguagem de máquina, através do

PROGRAMA 18

(6 bytes)

16514	2A	0C	40	LD HL,(16396)
16517	44			LD B,H } não existe LD BC,HL
16518	4D			LD C,L }
16519	C9			RET

PRINT USR 16514 fornece o endereço do primeiro byte do arquivo de imagem. Sabemos que este endereço contém 118, um NEW LINE. Assim, se você digitar

PRINT PEEK (PEEK 16396+256*PEEK 16397)

a resposta sempre será 118.

O esquema da página seguinte ajudá-lo-á a compreender a tela.

O número na quadrícula é o que deve ser *somado* ao endereço apontado por D.FILE para se obter o endereço da posição de tela.

Assim, observando o esquema, tente entender as próximas explicações.

- a) O conteúdo dos endereços relativos às quadrículas hachuradas é sempre 118, e não pode ser alterado, ou o micro "se perderia", sem saber "construir" a tela... CRASH. Veja e compreenda a saída deste programa BASIC.

```
10 LET DF=PEEK 16396+256*PEEK 16397
20 FOR L=0 TO 24
30 PRINT PEEK (DF+33*L),
40 NEXT L
```

Sua saída serão 25 números 118! Ou seja, o NEW LINE inicial do arquivo e mais os 24 NEW LINEs terminais de cada linha.

- b) Você pode imprimir na tela, fazendo POKE em posições permitidas da tela.

C. FILE		0	Linha 0
1	2	3	4
34		5	6
67		7	8
100		9	10
133		11	12
166		13	14
199		15	16
232		17	18
265		19	20
298		21	22
331		23	24
364		25	26
397		27	28
430		29	30
463		31	32
496		33	
529			
562			
595			
628			
661			
694	695	696	697
727	— LINHAS DE EDIÇÃO —	698	699
759	760	761	762

FIG. 1 — A tela de TV.

Observe os exemplos.

1º Exemplo: 10 LET DF=PEEK 16396+256*PEEK 16397
20 POKE DF+1,23

Isto imprime um asterisco na posição 0,0

2º Exemplo: 10 LET DF=PEEK 16396+256*PEEK 16397
20 LET A=23
30 POKE DF+1,A
40 POKE DF+32,A
50 POKE DF+694,A
60 POKE DF+725,A

Isto imprime quatro asteriscos, um em cada canto do vídeo.

3º Exemplo: Vamos imprimir um asterisco no meio da tela, ou seja, na posição 11,16. Como calcular o número que deve ser somado a DF? É fácil...

Chamando de L à linha e C à coluna; PRINT AT L,C;"**" equivale à POKE DF+33*L+C+1,23. Assim, para PRINT AT 11,16;"**" devemos somar a DF $33*11+16+1=380$.

Confira:

10 LET DF=PEEK 16396+256*PEEK 16397
20 PRINT AT 11,16;"N"
30 POKE DF+380,23

Só há um asterisco na tela, e a indicação 0/30. Se você prestar bastante atenção, ainda verá o N sendo impresso, e substituído pelo *.

4º Exemplo: Vamos fazer um X na tela usando 21 linhas. O X deve começar em 0,0 e 20,0 (pontos superiores) e terminar em 20,0 e 20,20 (pontos inferiores). O caractere de construção será *. Não passe adiante sem entender este programa!

10 LET DF=PEEK 16396+256*PEEK 16397
20 LET A=23
30 FOR S=1 TO 681 STEP 34
40 POKE DF+S,A
50 NEXT S
60 FOR S=21 TO 661 STEP 32
70 POKE DF+S,A
80 NEXT S

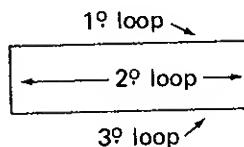
Observe bem. O loop das linhas 30 a 50 faz a perna \ do X. Por que STEP 34? O loop das linhas 60 a 80 faz a perna / do X. Por que STEP 32?

7.4

EXERCÍCIOS

- 54 Faça um programa BASIC semelhante aos que vimos nestes exemplos que preencha a 1ª linha da tela com *.
- 55 Faça um programa BASIC que preencha a última linha "normal" da tela (sem contar as linhas de edição) com *.

- 56 Faça um programa BASIC que preencha a 1^a coluna da tela, desde 0,0 até 21,0 com *.
- 57 Combinando tudo isto, faça um programa que coloque uma *moldura* na tela, usando apenas 3 loops. Observe o esquema:



Use * como caractere de construção da moldura. É muito importante que você faça este exercício, porque sua estrutura será necessária para que você compreenda quando o escrevermos em ASSEMBLY.

7.5

NOVAS INSTRUÇÕES ASSEMBLY

Vamos incrementar o conjunto de instruções que conhecemos, a fim de que possamos passar todos estes programas BASIC — interessantes, sem dúvida, mas *lentos* — para linguagem de máquina e sua conhecida velocidade.

1º Conjunto: UM POUCO MAIS DE LOAD (FAZENDO PEEK)

INSTRUÇÃO	CÓDIGO HEXA	CORRESPONDÊNCIA BASIC
LD A,(BC)	0A	LET A=PEEK BC
LD A,(DE)	1A	LET A=PEEK DE
LD A,(HL)	7E	LET A=PEEK HL (utilíssimo)
LD B,(HL)	46	LET B=PEEK HL
LD C,(HL)	4E	LET C=PEEK HL
LD D,(HL)	56	LET D=PEEK HL
LD E,(HL)	5E	LET E=PEEK HL
LD H,(HL)	66	LET H=PEEK HL
LD L,(HL)	6E	LET L=PEEK HL

2º Conjunto: UM POUCO MAIS DE LOAD (FAZENDO POKE)

INSTRUÇÃO	CÓDIGO HEXA	CORRESPONDÊNCIA BASIC
LD (BC),A	02	POKE BC,A
LD (DE),A	12	POKE DE,A
LD (HL),A	77	POKE HL,A (utilíssimo)
LD (HL),B	70	POKE HL,B

LD (HL),C	71	POKE HL,C
LD (HL),D	72	POKE HL,D
LD (HL),E	73	POKE HL,E
LD (HL),H	74	POKE HL,H
LD (HL),L	75	POKE HL,L
LD (HL),N	36	POKE HL,N (utilíssimo)

3º Conjunto: UM POUCO MAIS DE ADD

INSTRUÇÃO	CÓDIGO HEXA	EQUIVALÊNCIA BASIC
ADD A,(HL)	86	LET A=A+PEEK HL
ADD A,N	C6	LET A=A+N
ADD HL,BC	09	LET HL=HL+BC
ADD HL,DE	19	LET HL=HL+DE
ADD HL,HL	29	LET HL=HL+HL

4º Conjunto: UM POUCO MAIS DE INC E DEC

INSTRUÇÃO	CÓDIGO HEXA	EQUIVALÊNCIA BASIC
INC (HL)	34	LET PEEK HL=PEEK HL+1
INC BC	03	LET BC=BC+1
INC DE	13	LET DE=DE+1
INC HL	23	LET HL=HL+1
DEC (HL)	35	LET PEEK HL=PEEK HL-1
DEC BC	0B	LET BC=BC-1
DEC DE	1B	LET DE=DE-1
DEC HL	2B	LET HL=HL-1

IMPORTANTE:

Estas instruções NÃO afetam nenhuma FLAG, e não podem ser aproveitadas para contadores!

7.6

USANDO O ARQUIVO DE TELA

Vamos agora entrar em terreno muito fértil. Começamos o estudo de programas onde é usado o arquivo de tela. O limite é a sua imaginação. Podemos fazer o que desejarmos com a tela, é só criar...

Nosso primeiro programa será bem simples: colocar um asterisco bem no meio da tela (PRINT AT 11,16; "**) . Reveja como isto foi feito no 3º exemplo do item 7.3.

Nós faremos assim:

- a) LD HL,(16396) → faz de HL ponteiro para o arquivo de imagem; HL agora aponta para o NEW LINE inicial do arquivo.
- b) LD DE,380 → coloca em DE o número de posições a serem saltadas.
- c) ADD HL,DE → salta as posições necessárias.
- d) LD (HL),23 → coloca o asterisco na tela.
- e) RET.

Detalhes: 1º) $16396 = 400C$ (ver Apêndice 5); cuidado com a inversão!

2º) $380 = 017C$, pois

$380 | 256$
124 1
7C 01h

Agora estamos prontos.

PROGRAMA 19

(10 bytes)

Ø REM 10 caracteres quaisquer

16514	2A	0C	40	LD HL,(16396)
16517	11	7C	01	LD DE,380
16520	19			ADD HL,DE
16521	36	17		LD (HL),23
16523	C9			RET

Nosso segundo exemplo será produzir a linha superior da tela preenchida com *****.

Esquematizemos:

- a) LD HL,(16396).
- b) LD B,32 → faz de B um contador para 32 caracteres colocados na tela.
- c) INC HL → salta uma posição ←
- d) LD (HL),151
- e) DJNZ —————
- f) RET

Cuidado para calcular o salto relativo para trás.

PROGRAMA 20

(11 bytes)

Ø REM 11 caracteres quaisquer

16514	2A	ØC	40	LD HL,(16396)
16517	Ø6	20		LD B,32
16519	23			INC HL ←
16520	36	97		LD (HL),151
16522	10	FB		DJNZ -5
16524	C9			RET

Nosso terceiro exemplo será produzir uma diagonal da tela, desde Ø,Ø até 21,21. É fácil. Veja lá. Usaremos o caractere número 134, fica bonito.

PROGRAMA 21

(15 bytes)

16514	2A	ØC	40	LD HL,(16396)
16517	23			INC HL ; salta o NEW LINE inicial
16518	11	22	00	LD DE,34 ; nº de posições a saltar
16521	Ø6	16		LD B,22 ; contador de linhas
16523	36	86		LD (HL),134 ; coloca na tela ←
16525	19			ADD HL,DE ; salta 34 posições
16526	10	FB		DJNZ -5 ; volta para LOOP
16528	C9			RET

Nosso quarto exemplo será um programa para encher a tela com o caractere 8. Preste bastante atenção à estrutura do programa. É importante para que você possa compreender outros.

PROGRAMA 22

(17 bytes)

16514	2A	ØC	40	LD HL,(16396)
16517	ØE	16		LD C,22 ; contador de linhas
16519	23			INC HL ; salta NEW LINE ←
16520	Ø6	20		LD B,32 ; contador de colunas
16522	36	Ø8		LD (HL),8 ; coloca na tela ←
16524	23			INC HL ; próxima posição
16525	10	FB		DJNZ -5 ; completa linha
16527	ØD			DEC C ; decrementa contador de linhas
16528	20	F5		JR NZ, -11 ; enche a tela
16530	C9			RET

Este programa 22, levemente adaptado, produz um espetacular efeito de tela, que você pode usar como preferir em seus programas BASIC.

PROGRAMA 23

(21 bytes)

16514	3E	3F	LD A,63 ; inicializa caractere de impressão com Z
16516	2A	0C 40	LD HL,(16396) ←
16519	0E	16	LD C,22
16521	23		INC HL ←
16522	06	20	LD B,32
16524	77		LD (HL),A ←
16525	23		INC HL
16526	10	FC	DJNZ -4
16528	0D		DEC C
16529	20	F6	JR NZ,-10
16531	3D		DEC A
16532	20	EE	JR NZ,-18
16534	C9		RET

Que tal o efeito? Impressionante a velocidade, não? Bem, agora é com você...

7.7

EXERCÍCIOS

- 58 Coloque, usando a memória de tela, um ponto "final" na tela. Ou seja: PRINT AT 21,31; ". .".
- 59 Preencha a 1ª coluna com █, ou seja, usando a memória de tela, faça algo equivalente ao BASIC

```
10 FOR I=0 TO 21
20 PRINT " █ "
30 NEXT I
```

- 60 O Exercício 58 é semelhante ao Programa 19, o Exercício 59 tem a mesma estrutura que o Programa 21. Para você, terminaram as facilidades... Daqui para frente, vamos exigir muita criatividade de sua parte. Cada exercício desafiará você, de alguma maneira. Mas, não se preocupe:

- os exercícios estão em ordem de dificuldade crescente;
- se estamos juntos até agora, é porque você gosta do assunto pelo menos tanto quanto eu;
- você é capaz! Vamos lá!

Faça uma diagonal da tela, mas começando em 0,31 e terminando em 21,10, usando o caractere de código 6.

O programa BASIC equivalente seria

```
10 FOR I=0 TO 21
20 PRINT AT I,31-I;" █ "
30 NEXT I
```

- 61 Combinando idéias já vistas até agora, escreva um programa que encha a tela da seguinte forma:

linha 0 → caractere 0

linha 1 → caractere 1

linha 2 → caractere 2

:

:

:

:

linha 20 → caractere K

linha 21 → caractere L

Se você acertar, deverá gastar 19 bytes.

- 62 Vamos fazer três adaptações no Programa 23. A primeira delas é para que os caracteres que se sucedem na tela comecem no caractere número 1 e terminem em Z. Sugestão: initialize o caractere com 1, incremente-o e compare com 64.

- 63 A segunda é que os caracteres iniciem com Z e terminem no caractere número 136. Sugestão: initialize o caractere com .Z, decremente-o e compare com 135. Isto pode ser um bonito efeito de abertura de jogos...

- 64 A terceira é positivamente espetacular: como passar por todos os caracteres da linha Sinclair? Observe atentamente o Exercício 51, e a idéia lhe surgirá. Na verdade, o programa é escrito duas vezes, com a mesma estrutura, o elo de ligação está no Exercício 51. Você deve gastar 44 bytes! Já vi um programa numa revista especializada que faz o mesmo efeito, mas gastando 71 bytes e usando uma estrutura de loops tão complicada que era um verdadeiro quebra-cabeças seguir o programa... O programa que você vai fazer é econômico e simples, são virtudes.

RESPOSTAS DOS EXERCÍCIOS

54 10 LET DF=PEEK 16396+256*PEEK 16397

20 FOR S=1 TO 32

30 POKE DF+S,151

40 NEXT S

55 10 LET DF=PEEK 16396+256*PEEK 16397

20 FOR S=694 TO 725

30 POKE DF+S,151

40 NEXT S

56 10 LET DF=PEEK 16396+256*PEEK 16397

20 FOR S=1 TO 694 STEP 33

30 POKE DF+S,151

40 NEXT S

57 10 LET DF=PEEK 16396+256*PEEK 16397

20 LET A=151

30 FOR S=1 TO 32

40 POKE DF+S,A

50 NEXT S

60 FOR S = 34 TO 661 STEP 33
 70 POKE DF + S,A
 80 POKE DF + S + 31,A
 90 NEXT S
 100 FOR S=694 TO 725
 110 POKE DF + S,A
 120 NEXT S

58 Ø REM 10 caracteres quaisquer

16514	2A	0C	40	LD HL,(16396)
16517	11	D5	02	LD DE,725
16520	19			ADD HL,DE
16521	36	1B		LD (HL),27
16523	C9			RET

Duas maneiras para obter o número 725:

- 1º) $33 \times L + C + 1 = 33 \times 21 + 31 + 1 = 693 + 32 = 725$
- 2º) olhando no esquema...

$$\begin{array}{r}
 725 | 256 \\
 213 \quad 2 \\
 \swarrow \qquad \searrow \\
 D5 \qquad 02
 \end{array}$$

59 Ø REM 15 caracteres quaisquer

16514	2A	0C	40	LD HL,(16396)
16517	23			INC HL
16518	11	21	00	LD DE,33
16521	06	16		LD B,22
16523	36	97		LD (HL),151
16525	19			ADD HL,DE
16526	10	FB		DJNZ -5
16528	C9			RET

60 Ø REM 14 caracteres quaisquer

16514	2A	0C	40	LD HL,(16396)
16517	11	20	00	LD DE,32
16520	06	16		LD B,22
16522	19			ADD HL,DE
16523	36	06		LD (HL),6
16525	10	FB		DJNZ -5
16527	C9			RET

61 Ø REM 19 caracteres quaisquer

INF CIO	16514	2A	0C	40	LD HL,(D.FILE)
	16517	3E	1C		LD A,28 ; caractere Ø
	16519	0E	16		LD C,22 ; conta linhas
LOOP 1	16521	23			INC HL ; salta NEW LINE
	16522	06	20		LD B,32 ; conta colunas

	LOOP 2	16524	77		LD (HL),A ; coloca caractere na tela
		16525	23		INC HL ; próxima posição
		16526	10	FC	DJNZ LOOP 2; completa linha
		16528	3C		INC A ; próximo caractere
		16529	0D		DEC C ; decrementa contador de linhas
		16530	20	F5	JR NZ LOOP 1; próxima linha
		16532	C9		RET
62	INÍCIO	16514	3E	01	LD A,1
	TELA	16516	2A	0C	40 LD HL,(D.FILE)
		16519	0E	16	LD C;linhas
	LOOP 1	16521	23		INC HL
		16522	06	20	LD B,colunas
	LOOP 2	16524	77		LD (HL),A
		16525	23		INC HL
		16526	10	FC	DJNZ LOOP 2
		16528	0D		DEC C
		16529	20	F6	JR NZ,LOOP 1
		16531	3C		INC A
		16532	FE	40	CP 64
		16534	20	EC	JR NZ,TELA
		16536	C9		RET

O programa ocupa ao todo 23 bytes, colocamos alguns rótulos para melhorar a sua compreensão.

- 63 Se você realmente está entendendo, não encontrou a menor dificuldade neste exercício. Tem apenas três bytes diferentes em relação ao Exercício 62. No endereço 16515, coloque 191 (POKE 16515,191), que transforma a instrução de inicialização do caractere em 3E BF (LD A,191). No endereço 16528, coloque 61 (POKE 16531,61), que transforma a instrução de atualização do caractere em 3D (DEC A). Finalmente, no endereço 16533, coloque 135 (POKE 16533,135), que transforma a instrução de controle do caractere em FE 87 (CP 135). Digite RAND USR 16514. Após iniciar em e vir "descendo" a lista de caracteres, a tela pára cinza. Verdade? Vamos usar este efeito. Suponha que você escreveu um jogo bem animado, e quer fazer uma bonita introdução para ele. Que tal esta?

```

0 REM aqui está o programa do Exercício 63
5 PRINT AT 11,6;"DIGITE J PARA JOGAR"
10 IF INKEY$<>"" THEN GOTO 10
15 IF INKEY$="" THEN GOTO 15
20 LET I$=INKEY$
25 IF I$<>"J" THEN GOTO 10
30 RAND USR 16514
35 PRINT "TITULO DO JOGO"
etc.

```

Escreva o título do jogo *sobre* a tela cinza, fica bonito. E você tem uma abertura bem movimentada, como o seu jogo.

64 Ø REM 44 bytes

INÍCIO	16514	3E	Ø1		LD A,1
TELA	16516	2A	ØC	40	LD HL,(D.FILE)
	16519	ØE	16		LD C,linhas
LOOP 1	16521	23			INC HL
	16522	Ø6	2Ø		LD B,colunas
LOOP 2	16524	77			LD (HL),A
	16525	23			INC HL
	16526	1Ø	FC		DJNZ LOOP 2
	16528	ØD			DEC C
	16529	2Ø	F6		JR NZ,LOOP 1
	16531	3C			INC A
	16532	FE	4Ø		CP 64
	16534	2Ø	EC		JR NZ,TELA
INÍCIO'	16536	87			ADD A,A ; passa A de 64 para 128
TELA'	16537	2A	ØC	40	LD HL,(D.FILE)
	16540	ØE	16		LD C,linhas
LOOP 1'	16542	23			INC HL
	16543	Ø6	2Ø		LD B,colunas
LOOP 2'	16545	77			LD (HL),A
	16546	23			INC HL
	16547	1Ø	FC		DJNZ LOOP 2'
	16549	ØD			DEC C
	16550	2Ø	F6		JR NZ LOOP 1'
	16552	3C			INC A
	16553	FE	CØ		CP 192
	16555	2Ø	EC		JR NZ,TELA'
	16557	C9			RET

Em relação aos 71 bytes do programa que vi, me parece que o nosso está bem melhor...

8.1

INTRODUÇÃO

Neste capítulo vamos exercitar o que aprendemos. Veremos exercícios de todos os tipos: fáceis, médios, difíceis... O primeiro deles funcionará como uma verdadeira revisão.

8.2

EXERCÍCIOS

65 BOAS E MÁS IDÉIAS PARA ENCHER A TELA

Vamos agora a um longo exercício, dividido em duas partes. O exercício exigirá muito de você, em atenção e criatividade.

O programa que apresentamos funciona: ele enche a tela com o conteúdo do endereço 16514, e retorna ao BASIC. Acontece que ele funciona APESAR de todos os seus defeitos estruturais: o programa pode e deve ser *muito* melhorado.

Quem vai melhorá-lo? Ora, você, é claro.

1ª PARTE

Faça a desmontagem do programa, preenchendo as lacunas. Use e abuse do Apêndice 9. Observe que numeramos as instruções, para que depois você responda algumas coisas sobre elas.

16514	17			código do caractere
16515	2A	0C	40	inst. 1:.....
16518	23			inst. 2:.....
16519	06	16		inst. 3:.....
16521	0E	21		inst. 4:.....
16523	7E			inst. 5:.....
16524	FE	76		inst. 6:.....
16526	28	04		inst. 7:.....
16528	3A	82	40	inst. 8:.....
16531	77			inst. 9:.....
16532	23			inst. 10:.....
16533	0D			inst. 11:.....
16534	20	F3		inst. 12:.....
16536	10	EF		inst. 13:.....
16538	C9			inst. 14:.....

ATENÇÃO: RAND USR 16515

Agora que você fez a desmontagem do programa, responda:

- a) É necessária uma linha \emptyset REM com quantos caracteres?

.....

- b) No endereço 16514, está o código de qual caractere?

.....

- c) Qual a instrução 1? Para que serve?

.....

- d) Qual a diferença entre LD HL,NN e LD HL,(NN)?

.....

.....

- e) Qual a instrução 2? Para que serve?

.....

.....

- f) Os registradores de 16 bits são úteis como contadores? Por quê? HL está sendo usado como contador?

.....

.....

- g) Qual a instrução 3? Para que serve?

.....

.....

- h) Qual a instrução 4? Para que serve?

.....

.....

- i) Qual a instrução 5? Para que serve?

.....

.....

- j) Qual a instrução 6? Para que serve?

.....

.....

- k) Observando agora a instrução 6, a instrução 5 poderia ter usado outro registrador que não A (por exemplo D ou E, não usados neste programa)?

.....

.....

- l) Qual a instrução 7? Para que serve?

.....

.....

- m) No caso de ser implementada, para que endereço a instrução 7 desvia o programa?

.....

- n) Qual a instrução 8? Para que serve?

.....

o) Qual a instrução 9? Para que serve?

.....

p) Qual a instrução 10? Para que serve?

.....

q) Qual a instrução 11? Para que serve?

.....

r) Qual a instrução 12? Para que serve?

.....

s) No caso de ser implementada, para que endereço a instrução 12 desvia o programa?

.....

t) Qual a instrução 13? Para que serve?

.....

.....

u) No caso de ser implementada, para que endereço a instrução 13 desvia o programa?

.....

v) Qual a instrução 14? Para que serve?

.....

2ª PARTE

Vamos agora à fase mais difícil, mas também mais interessante do trabalho: otimizar o programa, a partir das sugestões que daremos.

1ª sugestão: Reservar o endereço 16514 para o código do caractere, carregar o acumulador com o conteúdo deste endereço e colocar o conteúdo do acumulador na tela, tudo isto consome 5 bytes. Senão veja:

1 byte para o código do caractere, colocado em 16514

3 bytes para LD A,(16514)

1 byte para LD (HL),A

Tudo isto pode ser reduzido a apenas 2 bytes, com o uso da instrução LD HL, caractere.

2ª sugestão: O final de linha está sendo verificado 2 vezes: pelo contador C e pela comparação com 118. Naturalmente *apenas uma* verificação é necessária. No Programa 22, usamos dois contadores: C para linhas e B para colunas. Aqui, para fazermos um programa diferente (no bom sentido, é claro), sugerimos a comparação com 118.

3ª sugestão: É possível colocar *apenas uma* instrução INC HL, ao invés das duas do programa.

Sugestão de roteiro para o programa otimizado

LD B,linhas

LD HL,(D.FILE)

LOOP

INC HL

	LD A,(HL)
	CP NEW LINE
	JR Z,PROXLINHA
	LD (HL),caractere
	JR,LOOP
PROXLINHA	DJNZ LOOP
	RET

Seu programa deverá gastar 18 bytes (7 bytes a menos do que o programa original). Escreva-o cuidadosamente, calcule os saltos, e rode-o. Após vê-lo funcionar, confira com nossa resposta.

A finalidade deste programa é basicamente didática — observar as várias maneiras de se chegar a uma mesma finalidade.

Estude cuidadosamente o programa que você fez, e não passe adiante enquanto não entender o funcionamento exato de cada instrução. Lembre-se de que o programa foi otimizado, e cada instrução nele presente é *realmente* necessária.

66 TROCANDO CARACTERES NA TELA

Este programa faz um efeito semelhante à inversão de vídeo, e vai nos preparar para entendê-la. Sim, porque inversão de vídeo, um efeito que você certamente já viu (e quem sabe até invejou) em programas comerciais, é um dos próximos que faremos! Ou melhor, você fará!

Esta rotina em linguagem de máquina troca todos os caracteres *espaço* (endereço 16524¹⁴) por *espaço invertido* (endereço 16528). Naturalmente, se lhe interessar, você pode, de acordo com suas conveniências, alterar os conteúdos destes endereços. Por exemplo, para trocar o ponto decimal americano pela vírgula decimal brasileira na apresentação de um programa financeiro, que envolva quantias *brasileiras* em dinheiro. Nem só para joguinhos e efeitos de tela serve a linguagem de máquina...¹⁵

Vamos lá, a rotina ocupa apenas 22 bytes.

	LD HL,(D.FILE)
	LD B,linhas
LINHA	INC HL
	LD C,colunas
PEEK	LD A,(HL)
	CP espaço
	JR NZ,PROXPOS
	LD (HL),espaço invertido
PROXPOS	INC HL
	DEC C
	JR NZ,PEEK
	DJNZ LINHA
	RET

Após escrever o programa em códigos hexa, *tomando o habitual cuidado com os saltos para trás e para frente*, digite (modo direto) RAND USR 16514. Se você acertou, deverá obter toda a tela cheia com espaço invertido. É isto a rotina? Não, claro.

Cerque-a com o seguinte programa BASIC, alterando a linha 10 para o PRINT AT que achar melhor...

```
10 PRINT AT 10,8;"NOME DO PROGRAMA"
20 GOSUB 100
30 RAND USR 16514
40 GOSUB 100
50 CLS
60 LIST
70 GOSUB 100
80 RAND USR 16514
90 STOP
100 FOR X=1 TO 50
110 NEXT X
120 RETURN
```

(Apague todas as linhas do CARREGADOR ASSEMBLY, para garantir que a listagem caiba na tela, do contrário seu programa parará com 5/60.)

67 A ESCADA

O programa abaixo enche a tela com dois caracteres (* e █), retornando ao BASIC sem problemas. O efeito é bonito, e muito veloz. Digite-o (você verá por que o nome "escada"), e estude detalhadamente cada instrução. Seu exercício está dividido em várias partes.

1ª PARTE: DESMONTAGEM DO PROGRAMA

Ø REM 33 caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS	COMENTÁRIOS
	16514	2A ØC 40		
	16517	23		
	16518	06 16		
	16520	0E 1F		
	16522	3E 17		
	16524	51		
	16525	1E 20		
	16527	77		
	16528	23		
	16529	1D		
	16530	15		
	16531	C2 8F 40		
	16534	3E 97		
	16536	77		
	16537	23		
	16538	1D		
	16539	C2 98 40		
	16542	23		
	16543	ØD		
	16544	10 E8		
	16546	C9		

2ª PARTE: ALTERAÇÕES

Bem, agora você está convidado a *alterar* este programa, reduzindo-o a apenas 28 bytes. Eis as nossas sugestões:

- 1º) O endereço indicado por HL pode ser carregado diretamente, sem o uso do registrador A, através de instrução LD (HL),N, cujo código hexa é 36 N, onde N é um número em hexa na faixa de 00 até FF. Assim, LD A,23 e LD (HL),A podem ser substituídos por LD (HL),23. Da mesma forma, LD A,151 e LD (HL),A podem ser substituídos por LD (HL),151. Isto economiza um byte de cada vez, ou seja, 2 bytes ao total.
- 2º) Transforme os saltos absolutos em saltos relativos, ou seja, troque JP NZ por JR NZ. Use a tabela de complemento de dois — Apêndice 4. Isto, além de economizar 1 byte de cada vez, ou seja, 2 bytes no total, faz com que o programa possa ser colocado a partir de *qualquer* endereço inicial, e não obrigatoriamente de 16514.
- 3º) Alterando a *ordem* das instruções iniciais, e o endereço de retorno da instrução DJNZ, você pode dispensar o INC HL atualmente no endereço 16537 e “aproveitar” o INC HL que está agora no endereço 16517 para a mesma finalidade. Frisando bem: para conseguir isto é necessário *alterar a ordem das instruções iniciais*. Isto economiza mais um byte.

Bom trabalho. Digite o *seu* programa, verifique se o efeito é o mesmo, e confira com a resposta que apresentamos.

3ª PARTE: MAIS ALTERAÇÕES

Seu exercício agora é alterar novamente o programa, diminuindo ainda mais o número de bytes, desta vez para *apenas* 24 bytes (!). A sugestão desta vez é usar a instrução RST 10 que, como você sabe, coloca na tela o conteúdo do acumulador (A). Assim sendo,

- dispense:
LD HL,(D.FILE)
INC HL
LD (HL),23
LD (HL),151
- aproveite: os contadores e o esquema de saltos relativos, naturalmente recalculados
- use:

{ LD A,23	para colocar * na tela
RST 10	

{ LD A,151	para colocar █ na tela
RST 10	

Divirta-se.

68 MOLDURA NA TELA

A rotina que vamos apresentar coloca uma moldura na tela, usando para isto o caractere █. Seu exercício será fazer a desmontagem desta rotina, e explicar a finalidade de cada instrução.

Além de útil, pois usando-se como linha de programa BASIC a moldura é colocada instantaneamente na tela (RAND USR 16514), a rotina está escrita da maneira mais didática possível. Introduza alguns rótulos para facilitar a sua compreensão.

Ø REM 31 caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS	COMENTÁRIOS
.....	16514	2A 0C 40
.....	16517	06 20
.....	16519	3E 97
.....	16521	23
.....	16522	77
.....	16523	10 FC
.....	16525	06 14
.....	16527	23
.....	16528	23
.....	16529	77
.....	16530	11 1F 00
.....	16533	19
.....	16534	77
.....	16535	10 F6
.....	16537	06 20
.....	16539	23
.....	16540	23
.....	16541	77
.....	16542	10 FC
.....	16544	C9

Após ter feito a desmontagem da rotina, responda às seguintes perguntas:

- Em que endereço está o código do caractere que faz a moldura?
.....
- Que comando pode ser digitado a partir do BASIC (modo direto) para que o caractere que faz a moldura passe a ser espaço reverso?
.....
- Qual o endereço do último byte do conjunto de instruções que faz a linha superior da moldura?
.....
- Quais os endereços dos bytes inicial e final do conjunto de instruções que faz os lados da moldura?
.....
- Quais os endereços dos bytes inicial e final do conjunto de instruções que faz a linha inferior da moldura?
.....
- Qual a finalidade da instrução do endereço 16527?
.....

g) Qual o endereço da instrução de mesma finalidade que a citada na letra f?
.....

h) Que aconteceria se estas instruções fossem suprimidas?
.....

Agora, um exercício de programação. A idéia é obter o mesmo resultado final, usando a instrução RST 10. Se você acertar, a nova rotina levará 29 bytes.

ROTEIRO

1) A LINHA SUPERIOR

- a) use B como contador para 32 caracteres
- b) carregue o acumulador com *
- c) coloque-o na tela com RST 10
- d) volte para completar a linha

2) AS VINTE LINHAS DO MEIO

- a) use B como contador para 20 linhas
- b) coloque * na tela com RST 10
- c) use C como contador para 30 espaços
- d) carregue o acumulador com espaço
- e) coloque espaço na tela com RST 10
- f) decremente C
- g) volte para completar os 30 espaços
- h) carregue o acumulador com *
- i) coloque * na tela com RST 10
- j) volte para completar as 20 linhas

3) A LINHA INFERIOR

- a) use B como contador para 32 caracteres
- b) coloque * na tela com RST 10
- c) volte para completar a linha
- d) não esqueça da famosa instrução para retornar ao BASIC

Após rodar o programa, responda: por que ele é *mais lento* (visivelmente) que o programa apresentado no exercício, se gasta 2 bytes a menos?

69 A INVERSÃO DE VÍDEO

A idéia da inversão de vídeo é bastante simples. Senão, observe:

- a) É necessário varrer a tela toda; logo, 2 contadores: usemos B para contar linhas e C para contar colunas.
- b) Após saltar o NEW LINE, colocamos o que estiver na tela em A: LD A,(HL), o que equivale em BASIC a LET A=PEEK HL.
- c) Adicionamos 128 a A, o que tem o seguinte efeito:
se A=23 então o novo valor de A é 23+128=151

se A=151 então o novo valor de A=151+128=279, o que não "cabe" em A; logo A conterá 279-256=23.

Observando o Apêndice 2, você compreenderá que * é transformado em █, e █ é transformado em *.

- d) Colocamos o valor de A atualizado na tela: LD (HL),A, o que equivale em BASIC a POKE HL,A; e passamos à próxima posição.
e) Decrementos de contadores e saltos para trás completam o programa, que tem então a seguinte estrutura:

INÍCIO	LD HL,(D.FILE)
	LD B,linhas
SALTA NL	INC HL
	LD C,colunas
PEEK	LD A,(HL)
	ADD A,128
	LD (HL),A
	INC HL
	DEC C
	JR NZ,PEEK
	DJNZ SALTA NL
	RET

Mãos à obra! Se você escrevê-lo direito, deverá ocupar apenas 19 bytes.

Para testá-lo, vamos escrever um programa BASIC.

```
10 FOR F=0 TO 21
20 PRINT AT F,F;"M—I—C—R—O"
30 NEXT F
40 GOSUB 100
50 CLS
60 LIST
70 GOSUB 100
80 STOP
100 FOR N=1 TO 7
110 RAND USR 16514
120 FOR A=1 TO 2
130 NEXT A
140 NEXT N
150 RETURN
```

A sub-rotina das linhas 100 a 140 produz o efeito da tela "piscar", bastante usado para simular explosões em joguinhos. As linhas 120 e 130 fazem um retardo que melhora o efeito.

Agora, um desafio a você: adaptar o programa ASSEMBLY que escreveu para inverter somente os caracteres realmente impressos, *não invertendo os espaços*. Ou seja, espaços continuam a ser espaços. Sugestão: após LD A,(HL), fazer CP 0 e o JR Z adequado para saltar ADD A,128 e LD (HL),A, passando para a próxima posição da tela ao encontrar o espaço. Isto provoca um aumento de 4 bytes: o novo programa deve gastar 23 bytes. Teste-o com o mesmo programa BASIC.

70 UM NOVO CLS

A idéia deste programa me surgiu através da leitura do livro já citado, *Programação em Assembler e Linguagem de Máquina*, de David C. Alexander, da Editora Campus. Ele é voltado para a linha TRS-80, que também usa o Z-80.

É possível "chamar" a sub-rotina da ROM que faz CLS. Consultando o "mapa da mina", ou melhor dizendo, o "mapa da ROM" do Apêndice 14, vemos que o endereço de CLS é 0A2A, ou seja, $10 \times 256 + 42 = 2602$.

Assim sendo, a digitação (modo direto) de RAND USR 2602 tem rigorosamente o mesmo efeito que CLS. Podemos fazer isto em um programa ASSEMBLY:

16514	CD	2A	0A	CALL 2602 (CALL CLS)
16517	C9			RET

Vamos escrever agora um programa NOSSO que faça CLS *mais depressa* que o CLS da ROM! Isto é possível? Claro!

Antes de mais nada, dois detalhes:

- 1º) Vamos colocá-lo imediatamente após o anterior, ou seja, do endereço 16518 em diante. Programas distintos em linguagem de máquina podem ser colocados numa mesma linha de REM, desde que você os "chame" no endereço correto. Veja lá.
 - para chamar o CLS da máquina, faremos RAND USR 16514;
 - para chamar o "nossa" CLS, RAND USR 16518.
- 2º) CLS não apenas limpa a tela, mas também posiciona o cursor de impressão em 0, 0. Observe como será a coisa:
 - endereços de 16518 até 16533, um programa do tipo do PROGRAMA 22 que encha a tela com espaços;
 - endereços de 16534 até 16536, CALL PRINT AT 0,0. Se não se lembrar mais, consulte o Capítulo 4.
 - endereço 16537, RET

Os dois programas ocupam juntos 24 bytes. Para que você possa constatar o quanto "nossa" programa é *mais rápido* do que o CLS do micro, digite este programinha BASIC.

```
10 GOSUB 100
20 RAND USR 16514
30 GOSUB 200
40 FOR T=1 TO 60
50 NEXT T
60 GOSUB 100
70 RAND USR 16518
80 GOSUB 200
90 STOP
100 FOR F=0 TO 21
110 PRINT AT F,F;"COMPUTADOR"
120 NEXT F
130 RETURN
200 PRINT "TELA LIMPA"
210 RETURN
```

INSTRUÇÕES QUE USAM A PILHA

O próximo programa — espetacular sob todos os pontos de vista — usa todos os registradores: A, B, C, D, E, H e L, é não é suficiente... Assim, para torná-lo possível, usaremos a PILHA DE DADOS (STACK), uma maneira excelente para “salvar” dados.

PILHA é a região da memória *após* o programa, e que está disponível para nós. A pilha é construída de trás para frente: seu valor inicial é dado pela variável do sistema ERR.SP, cujos endereços são 16386 e 16387.

Supónha que o endereço do primeiro byte disponível, para nós, da pilha seja 31000 (isto será apontado por ERR.SP). Vamos colocar na pilha os números 14, 29 e 35, nesta ordem.

ENDEREÇO	CONTEÚDO	PONTEIRO
31000	14	ERR.SP
30999	29	
30998	35	SP
30997	

Enquanto a variável do sistema ERR.SP indica a parte de baixo da pilha, um registrador especial de 16 bits, SP (stack pointer), aponta para o topo da pilha.

INSISTIMOS: a pilha é construída de trás para frente, ou de cabeça para baixo (visualize como achar melhor...).

Para retirar os números que colocamos lá, o primeiro que vai sair é 35, depois 29, depois 14. Este esquema — o último a entrar é o primeiro a sair — é conhecido pela expressão inglesa LIFO— last in, first out.

Podemos colocar na pilha os conteúdos dos registradores de 16 bits — a instrução chama-se PUSH. Ao colocarmos mais dados na pilha, o valor de SP é *decrementado*.

Podemos colocar no topo da pilha os conteúdos de BC, DE, HL e AF. Atenção especial para este par — é formado pelo acumulador e pelo registrador de estado F. Não existe nenhuma instrução para carregar (LD) AF.

Para retirar da pilha, a instrução é POP.

Os códigos hexa destas instruções são:

PUSH BC	C5	POP BC	C1
PUSH DE	D5	POP DE	D1
PUSH HL	E5	POP HL	E1
PUSH AF	F5	POP AF	F1

A correspondência BASIC é a seguinte:

PUSH HL
(coloca na pilha os conteúdos de H e L)

POKE SP-1,H

POKE SP-2,L

LET SP=SP-2

POP HL
(recupera da pilha os conteúdos de L e H *inalterados*)

LET L=PEEK SP

LET H=PEEK (SP+1)

LET SP=SP+2

Algumas observações:

- após PUSH BC e PUSH DE, para recuperar os valores a ordem correta é POP DE seguido de POP BC;
- se após PUSH BC e PUSH DE fazemos POP BC e POP DE, os conteúdos de BC e DE são *trocados*;
- o número de PUSH e POP numa sub-rotina deve ser igual.
- usaremos, no programa a seguir, PUSH AF e POP AF para salvar e depois recuperar o valor contido no acumulador.

8.4

UM ÚLTIMO EXERCÍCIO: O ESPELHO

- 71 O leitor já conhece nossa admiração por Lewis Carroll (ver Capítulo 5, Exercício 48). Vem dele, e novamente de *Through the Looking-Glass* (1871), a inspiração para o último programa deste capítulo. A passagem que citamos é intrigante:
- Você gostaria de viver no País dos Espelhos, Kitty? Será que lá lhe dariam leite? Talvez o leite do País dos Espelhos não seja bom para beber.

Nossa tarefa é obter uma rotina em linguagem de máquina que simule o efeito de um espelho colocado verticalmente no meio da tela de TV: coloque o caractere da posição a,b na posição a,31-b. Observe a Figura 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
*	*	*	E	D	I	T	O	R	A	*	*	*																			
										*	*	*	C	A	M	P	U	S	*	*	*										

espelho

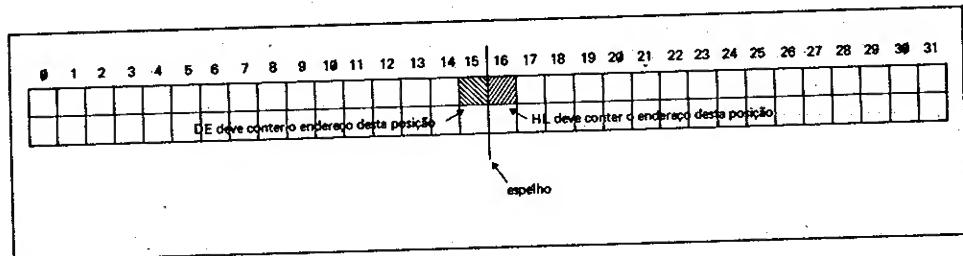
Se a tela inicial contiver isto, a rotina de máquina deverá convertê-la nisto

								*	*	*			S	U	P	M	A	C	*	*	*	A	R	O	T	I	D	E	*	*	*	

espelho

Como conseguir tal efeito? É simples e elegante. E incrivelmente econômico — a rotina inteira leva apenas 28 bytes. Vamos lá. Não ligue o micro ainda — primeiro vamos entender.

Vamos usar a tela inteira, e nosso costumeiro esquema de 2 contadores: B contará as linhas e C as colunas, mas desta vez C vai ser usado de maneira *diferente*. Para iniciar a rotina, precisamos de *dois* ponteiros para as posições de tela. Usaremos HL e DE. Desejamos que HL aponte para a primeira posição da primeira linha *após* o espelho, e DE aponte para a primeira posição da primeira linha *antes* do espelho. Veja a Figura 2.



Vejamos como conseguir isto:

LD HL,(16396)

LD DE,16

ADD HL,DE

HL agora aponta para a posição que desejamos para DE! Se não compreendeu, observe atentamente a figura da tela de TV do Capítulo 6, e reveja o Programa 19 e o Exercício 58. Se você andou saltando coisas por aí, vai ser difícil entender este programa...

Logo, precisamos transferir este endereço para DE:

LD D,H

LD E,L

É o jeito, pois não existe LD DE,HL...

Agora vamos fazer HL assumir o seu lugar:

INC HL

Chegamos à situação da Figura 2.

Agora precisamos trocar os conteúdos destes endereços e recolocá-los na tela trocados. B e C já estão comprometidos, pois serão usados como contadores. Assim, só nos resta o acumulador. Precisamos da pilha. Veja como será feito:

LD A,(HL)

Como sabemos, isto equivale a LET A=PEEK HL. O acumulador agora contém o código do caractere apontado por HL. Vamos guardar isto na pilha:

PUSH AF

Vamos agora apanhar o código do caractere apontado por DE

LD A,(DE)

e colocá-lo na posição apontada por HL

LD (HL),A

Isto equivale ao esquema BASIC:

LET A=PEEK DE
POKE HL,A

Agora vamos recuperar o código do caractere apontado por HL que está guardado na pilha:

POP AF

e colocá-lo no endereço apontado por DE

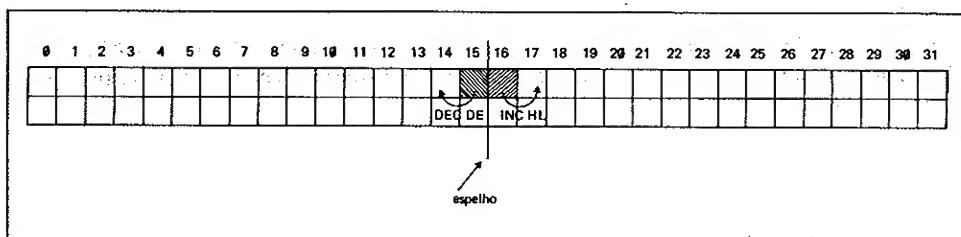
LD (DE),A

Está completa a troca! Para passarmos para as próximas posições da tela, guardando a idéia de espelho,

INC HL

DEC DE

Veja a Figura 3.



Quantas vezes precisamos repetir isto? Basta consultar a figura e ver que são 16 vezes para completar a linha.

Fazendo este loop, usando C como contador para as 16 trocas, chegaremos ao final da primeira linha, e a um ponto crucial do programa: *HL aponta agora para o NEW LINE do fim da primeira linha*. Ou seja, uma linha "mais tarde", estamos como estávamos após a instrução LD HL,(16396). Usaremos B como contador para 22 linhas, e está pronto o programa! Estude-o cuidadosamente, escreva-o em códigos hexa, com auxílio dos Apêndices 9 e 4, e rode-o!

Ø REM caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS	COMENTÁRIOS
INÍCIO	16514 16517	LD HL,(16396) LD B,22	; contador para 22 linhas.
PONTEIROS	16519 16522 16523	LD DE,16 ADD HL,DE LD D,H	

	16524	LD E,L	
	16525	INC HL	
	16526	LD C,16	
TROCA	16528	LD A,(HL)	; contador para 16 trocas por linha
	16529	PUSH AF	
	16530	LD A,(DE)	
	16531	LD (HL),A	
	16532	POP AF	
	16533	LD (DE),A	
	16534	INC HL	
	16535	DEC DE	
	16536	DEC C	
	16537	JR NZ,TROCA	
	16539	DJNZ PONTEIROS	
	16541	RET	

Para testar a rotina, de efeito totalmente inesperado, use o seguinte programa BASIC ("apague" as linhas do carregador ASSEMBLY, para a listagem "caber" em uma tela, pois iremos usá-la como parte do efeito).

```

10 INPUT I$
20 IF PEEK 16442=2 THEN CLS
30 PRINT I$
40 GOSUB 200
50 RAND USR 16514
60 GOTO 10
200 FOR T=0 TO 30
210 NEXT T
220 RETURN

```

Depois de brincar de inverter o seu nome e o de toda a família (o meu é NOSLEN e o de minha mulher é AIMEON...), acrescente estas linhas — o efeito é espetacular.

```

100 LIST
110 GOSUB 200
120 RAND USR 16514
130 GOTO 110

```

Digite RUN 100 e veja algo que nunca viu antes — nem poderia ver de outra maneira senão graças à linguagem de máquina...

RESPOSTAS DOS EXERCÍCIOS

65 DESMONTAGEM DO PROGRAMA

16514	23
16515	LD HL,(16396)
16518	INC HL

LINHA	16519	LD B,22
PEEK	16521	LD C,33
	16523	LD A,(HL)
	16524	CP 118
	16526	JR Z,4 (JR Z,PROXPOS)
	16528	LD A,(16514)
	16531	LD (HL),A
PROXPOS	16532	INC HL
	16533	DEC C
	16534	JR NZ,-13 (JR NZ, PEEK)
	16536	DJNZ-17 (DJNZ LINHA)
	16538	RET

(incluímos alguns rótulos para facilitar a compreensão).

RESPOSTAS DAS PERGUNTAS

- a) 25 caracteres quaisquer
- b) *
- c) LD HL,(16396) equivale a $\begin{cases} \text{LET L=PEEK 16396} \\ \text{LET H=PEEK 16397} \end{cases}$
Ou seja, o par de registradores HL é carregado com PEEK 16396 + 256 * PEEK 16397. Como sabemos, em 16396 (400C) "mora" a variável do sistema D.FILE, que contém o endereço do primeiro byte do arquivo de imagem.
- d) LD HL,NN carrega o par de registradores HL com o número representado por NN (um número entre 0 e 65535). LD HL,(NN) é equivalente a LET L=PEEK NN e LET H=PEEK (NN+1).
- e) INC HL. HL é agora o ponteiro do arquivo de tela, e seu primeiro "caractere" é um NEW LINE; que deve ser saltado. Lembre-se: é sempre verdade que $\text{PEEK } (\text{PEEK 16396} + 256 * \text{PEEK 16397}) = 118$.
- f) Não, porque as instruções INC e DEC para pares de registradores NÃO afetam nenhuma FLAG. Logo, como controlar o final da contagem? Não.
- g) LD B,22. Faz do registrador B um contador para 22 linhas. Lembre-se que B é privilegiado como contador, pela existência da instrução DJNZ.
- h) LD C,33. Faz do registrador C um contador para 33 "caracteres" por linha: os 32 da tela e um NEW LINE de final de linha.
- i) LD A,(HL). Coloca no registrador A o código do "caractere" apontado por HL. Equivale a LET A=PEEK HL.
- j) CP 118. Testa se o conteúdo de A é um caractere de vídeo ou um NEW LINE (118).
- k) Não, pois a instrução CP (COMPARE) tem como "alvo" exclusivo o registrador A, que é o acumulador.
- l) JR Z,4. No caso do caractere ser um NEW LINE ($118 - 118 = 0$ e a ZERO FLAG é ativada—"setada"), o programa é desviado para o endereço 16532 (rótulo PROXPOS-próxima posição), que contém a instrução INC HL.

- m) 16532.
- n) LD A,(16514). Carrega o acumulador com o conteúdo do endereço 16514 (o código do sinal gráfico *). Equivale a LET A=PEEK 16514.
- o) LD (HL),A. Coloca no vídeo o caractere *, cujo código foi transferido do endereço 16514 para o acumulador. Equivale a POKE HL,A; ou seja, POKE HL,23. Compare com a resposta da letra i.
- p) INC HL. Aponta para a próxima posição da tela.
- q) DEC C. Decrementa o contador C.
- r) JR NZ,-13. Enquanto o contador C não atingir 0 (fim de linha), o programa é desviado para o endereço 16523 (rótulo PEEK), que contém a instrução LD A,(HL).
- s) 16523.
- t) DJNZ -17. Decrementa o contador B, e enquanto não atingir 0 (fim de programa, pois teremos trabalhado as 22 linhas), o programa é desviado para o endereço 16521 (rótulo LINHA), que contém a instrução LD C,33. Ou seja, prepara para iniciar mais uma linha.
- u) 16521.
- v) RET. Volta ao BASIC.

O NOVO PROGRAMA

Ø REM 18 caracteres quaisquer

	16514	06	16	LD B,22
	16516	2A	0C	LD HL,(16396)
LOOP	16519	23		INC HL
	16520	7E		LD A,(HL)
	16521	FE	76	CP 118
	16523	28	04	JR Z,4 (JR Z,PROXLINHA)
	16525	36	17	LD (HL),23
	16527	18	F6	JR,-10 (JR,LOOP)
PROXLINHA	16529	10	F4	DJNZ -12 (DJNZ LOOP)
	16531	C9		RET

RAND-USR 16514

Trabalhe em cima deste programa até ter a certeza de tê-lo compreendido — tem uma estrutura bastante interessante.

66 Direto ao assunto.:

Ø REM 22 caracteres

	16514	2A	0C	40	LD HL,(16396)
	16517	06	16		LD B,22
LINHA	16519	23			INC HL
	16520	0E	20		LD C,32
PEEK	16522	7E			LD A,(HL)
	16523	FE	00		CP 0
	16525	20	02		JR NZ,2 (JR NZ,PROXPOS)

PROXPOS	16527	36	80	LD (HL),128
	16529	23		INC HL
	16530	0D		DEC C
	16531	20	F5	JR NZ,-11 (JR NZ, PEEK)
	16533	10	F0	DJNZ -16 (DJNZ LINHA)
	16535	C9		RET

Que tal o efeito? Observou a listagem como ficou? A listagem também é uma tela, suscetível a tudo que fizermos... .

67 1ª PARTE – O PROGRAMA COMO ERA

RÓTULOS INÍCIO	ENDEREÇOS	CÓDIGOS HEXA			MNEMÔNICOS	COMENTÁRIOS
LOOP	16514	2A	0C	40	LD HL,(16396)	; HL é ponteiro do arquivo de imagem
	16517	23			INC HL	; salta NEW LINE
	16518	06	16		LD B,22	; B conta linhas
	16520	0E	1F		LD C,31	; C conta *
	16522	3E	17		LD A,23	; A tem o código de *
	16524	51			LD D,C	; preserva o valor de C
IMPRIME 1	16525	1E	20		LD E,32	; E conta caracteres na linha
	16527	77			LD (HL),A	; coloca * na tela
	16528	23			INC HL	; próxima posição
	16529	1D			DEC E	; decrementa contador E
	16530	15			DEC D	; decrementa contador D
	16531	C2	8F	40	JP NZ,16527	; volta para IMPRIME 1
IMPRIME 2	16534	3E	97		LD A,151	; A tem o código de *
	16536	77			LD (HL),A	; coloca * na tela
	16537	23			INC HL	; próxima posição
	16538	1D			DEC E	; decrementa contador E
	16539	C2	98	40	JP NZ,16536	; volta para IMPRIME 2
	16542	23			INC HL	; salta o NEW LINE
	16543	0D			DEC C	; decrementa o contador C
	16544	10	E8		DJNZ -24	; volta para LOOP
	16546	C9			RET	; volta ao BASIC

2ª PARTE – O PROGRAMA ALTERADO

Ø REM 28 caracteres quaisquer

INÍCIO	16514	2A	0C	40	LD HL,(D.FILE)
	16517	06	16		LD B,22
	16519	0E	1F		LD C,31
LOOP	16521	23			INC HL
	16522	51			LD D,C
	16523	1E	20		LD E,32
IMPRIME 1	16525	36	17		LD (HL),23
	16527	23			INC HL
	16528	1D			DEC E
	16529	15			DEC D
	16530	20	F9		JR NZ,-7 (JR NZ,IMPRIME 1)
IMPRIME 2	16532	36	97		LD (HL),151
	16534	23			INC HL
	16535	1D			DEC E
	16536	20	FA		JR NZ,-6 (JR NZ,IMPRIME 2)
	16538	0D			DEC C
	16539	10	EC		DJNZ -20 (DJNZ,LOOP)
	16541	C9			RET

3ª PARTE – O PROGRAMA COM RST 10

Ø REM 24 caracteres quaisquer

INÍCIO	16514	06	16		LD B,22
	16516	0E	1F		LD C,31
LOOP	16518	51			LD D,C
	16519	1E	20		LD E,32
	16521	3E	17		LD A,23
IMPRIME 1	16523	D7			RST 10
	16524	1D			DEC E
	16525	15			DEC D
	16526	20	FB		JR NZ,-5 (JR NZ,IMPRIME 1)
	16528	3E	97		LD A,151
IMPRIME 2	16530	D7			RST 10
	16531	1D			DEC E
	16532	20	FC		JR NZ,-4 (JR NZ,IMPRIME 2)
	16534	0D			DEC C
	16535	10	ED		DJNZ -19 (DJNZ LOOP)
	16537	C9			RET

68 Ø REM 31 caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS	COMENTÁRIOS
INÍCIO	16514	2A	0C	LD HL,(16396)
	16517	06	20	LD B,32
	16519	3E	97	LD A,151 ; B= contador de caracteres na linha ; A = *

PROXPOS 1	16521	23		INC HL	; próxima posição na tela
	16522	77		LD (HL),A	; coloca * na tela
	16523	10	FC	DJNZ -4	; completar a linha superior
	16525	06	14	LD B,20	; B= contador de linhas
PROX LIN	16527	23		INC HL	; salta NEW LINE
	16528	23		INC HL	; próxima posição
	16529	77		LD (HL),A	; coloca * na tela
	16530	11	1F	40	LD DE,31
	16533	19		ADD HL,DE	; extensão do salto
	16534	77		LD (HL),A	; salta 31 posições
PROX POS 2	16535	10	F6	DJNZ -10	; colocar * na tela
	16537	06	20	LD B,32	; terminar os lados
	16539	23		INC HL	; B= contador de caracteres na linha
	16540	23		INC HL	; salta NEW LINE
	16541	77		LD (HL),A	; próxima posição
	16542	10	FC	DJNZ -4	; colocar * na tela
	16544	C9		RET	; completar a linha inferior

Vamos agora responder às perguntas:

- a) 16520
- b) POKE 16520,128
- c) 16524
- d) 16525 e 16536
- e) 16537 e 16543
- f) saltar o NEW LINE de fim de linha
- g) 16539
- h) Haveria CRASH do sistema, pois um caractere * seria colocado no lugar do NEW LINE de fim de linha, e o micro "se perderia", sem "saber" construir a tela.

Agora, o programa alternativo:

0 REM 29 caracteres quaisquer.

RÓTULOS INÍCIO	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS	COMENTÁRIOS
	16514	06 20	LD B,32	; B= contador de caracteres na linha
	16516	3E 97	LD A,151	; coloca * no acumulador

IMPRIME 1	16518	D7	RST 10	; coloca █ na tela
	16519	10 FD	DJNZ -3	; volta para completar a linha
	16521	06 14	LD B,20	; B= contador de linhas dos "lados"
IMPRIME 2	16523	D7	RST 10	; coloca █ na tela
	16524	0E 1E	LD C,30	; contador para 30 espaços
	16526	3E 00	LD A,0	; coloca espaço no acumulador
IMPRIME 3	16528	D7	RST 10	; coloca espaço na tela
	16529	0D	DEC C	; decrementa contador
	16530	20 FC	JR NZ,-4	; volta para completar 30 espaços
IMPRIME 4	16532	3E 97	LD A,151	; carrega acumulador com █
	16534	D7	RST 10	; completa a linha
	16535	10 F2	DJNZ -14	; volta para completar 20 linhas
	16537	06 20	LD B,32	; B= contador de caracteres na linha
	16539	D7	RST 10	; imprime █
	16540	10 FD	DJNZ -3	; volta para completar a linha
	16542	C9	RET	; volta ao BASIC

Este programa é mais lento do que o outro por dois motivos principais:

- RST 10 É MAIS LENTO do que fazer POKE direto na memória de vídeo. Lembre-se que RST 10 vai buscar uma sub-rotina da ROM para impressão. Isto perde tempo...
- Este programa faz 704 impressões, das quais 600 são espaços, e isto perde tempo. O programa do texto faz "apenas" 104 impressões ($104 = 32 + 20 + 20 + 32$) – tem de ser mais rápido. Tem-se a impressão de que a moldura "entra" instantaneamente na tela (desculpe o trocadilho...)

69 Ø REM 19 caracteres quaisquer

INÍCIO	16514	2A	0C	40	LD HL,(16396)
	16517	06	16		LD B,22
SALTA NL	16519	23			INC HL

	16520	0E	20	LD C,32
PEEK	16522	7E		LD A,(HL)
	16523	C6	80	ADD A,128
	16525	77		LD (HL),A
	16526	23		INC HL
	16527	0D		DEC C
	16528	20	F8	JR NZ,-8 (JR NZ, PEEK)
	16530	10	F3	DJNZ -13 (DJNZ SALTA NL)
	16532	C9		RET

Já vi escrito em livro sobre o assunto que uma rotina semelhante a esta "inverte os caracteres presentes na tela, desde que eles estejam no intervalo entre 0 e 63" (o grifo é nosso). Naturalmente que esta restrição NÃO EXISTE! A instrução ADD A,128 funciona da maneira que explicamos no texto do exercício.

Agora, o programa que "respeita" os espaços.

0 REM 23 caracteres quaisquer

INÍCIO	16514	2A	0C	40
	16517	06	16	
SALTA NL	16519	23		
	16520	0E	20	
PEEK	16522	7E		
	16523	FE	00	CP 0
	16525	28	03	JR Z,3 (JR Z,PROX POS)
	16527	C6	80	
	16529	77		
PROX POS	16530	23		INC HL
	16531	0D		
	16532	20	F4	JR NZ,-12 (JR NZ,PEEK)
	16534	10	EF	DJNZ -17 (DJNZ SALTA NL)
	16536	C9		

Naturalmente, alterando o conteúdo do endereço 16524, você pode programar o micro para inverter o vídeo "respeitando" qualquer caractere que você deseje...

7º 1º PROGRAMA

16514	CD	2A	0A	CALL 2602 (CALL CLS)
16517	C9			RET

2º PROGRAMA

16518	06	16		LD B,32
16520	2A	0C	40	LD HL,(16396)
16523	23			INC HL
16524	0E	20		LD C,32
16526	36	00		LD (HL),0
16528	23			INC HL
16529	0D			DEC C
16530	20	FA		JR NZ,-6
16532	10	F5		DJNZ -11

Note que nesta ocasião a tela está limpa, $B = \emptyset$ e $C = \emptyset$. Tudo perfeito para chamarmos a rotina de posicionamento de cursor (PRINT AT \emptyset,\emptyset). Para qualquer dúvida, recomendo que leia outra vez o Capítulo 4.

```
16534 CD F5 08 CALL 2293
16537 C9 RET
```

Os dois programas juntam-se e ocupam agora 24 bytes apenas.

Rode o programa BASIC fornecido, e aprecie a diferença de velocidades.

71 \emptyset REM 28 caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA			MNEMÔNICOS
INÍCIO	16514	2A	0C	40	LD HL,(16396)
	16517	06	16		LD B,22
PONTEIROS	16519	11	10	00	LD DE,16
	16522	19			ADD HL,DE
	16523	54			LD D,H
	16524	5D			LD E,L
	16525	23			INC HL
TROCA	16526	0E	10		LD C,16
	16528	7E			LD A,(HL)
	16529	F5			PUSH AF
	16530	1A			LD A,(DE)
	16531	77			LD (HL),A
	16532	F1			POP AF
	16533	12			LD (DE),A
	16534	23			INC HL
	16535	1B			DEC DE
	16536	0D			DEC C
	16537	20	F5		JR NZ,-11
	16539	10	EA		DJNZ -22
	16541	C9			RET

9.1

INTRODUÇÃO

Neste capítulo, vamos analisar um superprograma: o CLS ESPIRAL. É um programa relativamente longo, mas muito bem estruturado. Precisamos nos preparar para compreendê-lo.

9.2

FAZENDO PAUSA EM LINGUAGEM DE MÁQUINA

Uma das maneiras de fazer pausa em linguagem de máquina é usar um loop que não faz nada (existem outras maneiras, mas esta é a mais simples).

Em BASIC isto seria assim:

```
1000 FOR T=1 TO 100  
1010 NEXT T
```

Estas duas linhas criam uma pausa de aproximadamente 3 segundos. Este esquema pode ser usado como sub-rotina. Veja:

```
.....  
100 GOSUB 1000  
.....  
1000 FOR T=1 TO 100  
1010 NEXT T  
1020 RETURN  
.....
```

Em ASSEMBLY isto poderia ser feito da seguinte maneira:

```
.....  
PUSH AF (preserva o valor contido no acumulador)  
LD A,0  
DEC A } gasta tempo  
JR NZ,-3  
POP AF (recupera o valor contido no acumulador)  
.....
```

A pausa produzida é pequena, mas serve para nossos propósitos. Vamos verificar o efeito da introdução da pausa neste exemplo.

PROGRAMA 24

(19 bytes)

Ø REM 19 caracteres quaisquer

16514	2A	0C	40	LD HL,(16396)
16517	3E	80		LD A,128
16519	06	20		LD B,32
16521	23			INC HL
16522	77			LD (HL),A
16523	F5			PUSH AF
16524	3E	00		LD A,0
16526	3D			DEC A
16527	20	FD		JR NZ,-3
16529	F1			POP AF
16530	10	F5		DJNZ -11
16532	C9			RET

Como sugerimos em relação ao BASIC, também podemos fazer isto como uma sub-rotina, colocada no final do programa. E, em vez de GOSUB, CALL! Vamos combinar a idéia da sub-rotina com um efeito interessante: andar para trás na tela.

PROGRAMA 25

(27 bytes)

Ø REM 27 caracteres quaisquer

INÍCIO	16514	2A	0C	40	LD HL,(16396)
	16517	3E	80		LD A,128
	16519	06	20		LD B,32
	16521	11	D5	02	LD DE,725
	16524	19			ADD HL,DE
IMPRIME	16525	77			LD (HL),A
	16526	CD	95	40	CALL 16533 (CALL PAUSA)
	16529	2B			DEC HL
	16530	10	F9		DJNZ -7 (DJNZ IMPRIME)
	16532	C9			RET
PAUSA	16533	F5			PUSH AF
	16534	3E	00		LD A,0
	16536	3D			DEC A
	16537	20	FD		JR NZ,-3
	16539	F1			POP AF
	16540	C9			RET

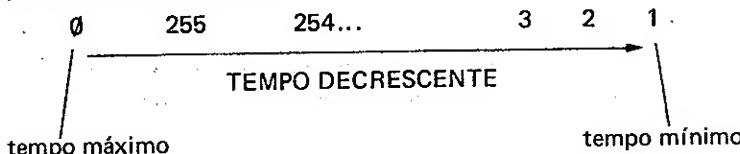
9.3

EXERCÍCIOS

- 72 Esta simples idéia de pausa cria alguns efeitos espetaculares. Faça a chamada de uma sub-rotina de pausa no local apropriado, e coloque esta sub-rotina no final do programa de INVERSÃO DE VÍDEO. Teste o efeito — é muito bonito — com este programa BASIC (não apague o carregador ASSEMBLY):

1000 LIST
 1010 RAND USR 16514
 1020 GOTO 1010

- 73 Introduza uma sub-rotina de pausa no final do programa "A ESCADA", alterado pro você para 28 bytes, e chame-a do local (*ou locais*) apropriado. O programa, que sempre gerou um bonito efeito, torna-se ainda mais bonito. Descubra o tempo de pausa mais adequado (a seu gosto), fazendo POKE direto. O tempo de duração da pausa é decrescente, assim:



- 74 A idéia de programar chamando sub-rotinas também em ASSEMBLY confere grande flexibilidade. Sugerimos aqui o esquema completo de um programa que pode ser usado como um "CLS HORIZONTAL". O efeito não é tão bonito quanto o do CLS ESPIRAL do fim do capítulo, mas a estruturação do programa é semelhante, e nos ajudará a entender o CLS ESPIRAL.

Prepara os códigos hexa deste programa:

Ø REM caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS
INÍCIO	16514	LD A,128 CALL ENCHE TELA
	LD A,0
	CALL ENCHE TELA
	CALL PRINT AT (2293)
	RET
ENCHE TELA	LD HL,(D.FILE)
	LD C,linhas
	INC HL
SALTA NL	LD B,colunas
	LD (HL),A
IMPRIME	CALL PAUSA
	INC HL
	DJNZ IMPRIME
	DEC C
	JR NZ,SALTA NL
PAUSA	RET
	PUSH AF
	LD A,0
	DEC A
	JR NZ,-3
	POP AF
	RET

Teste o CLS HORIZONTAL com o seguinte programa BASIC

```
1000 FOR F=0 TO 21
1010 PRINT AT F,F;"COMPUTADOR"
1020 NEXT F
1030 RAND USR 16514
1040 LIST
1050 RAND USR 16514
1060 PRINT "TELA LIMPA"
```

Se achar que o efeito está lento, dê um POKE no endereço correto e... acelere!

Estude cuidadosamente a estrutura deste programa — ele vai nos permitir entender o CLS ESPIRAL. As primeiras 5 instruções são a alma do programa.

9.4

MAIS UMA INSTRUÇÃO

Quando queríamos saltar posições na tela, usávamos ADD HL,BC ou ADD HL,DE. E se quisermos saltar para trás, é possível?

Naturalmente a resposta é sim. Usaremos a instrução SBC, mnemônico de SUBTRACT WITH CARRY. Aqui estão os códigos hexa:

SBC HL,BC	ED 42
SBC HL,DE	ED 52
SBC HL,HL	ED 62
SBC HL,SP	ED 72

Vamos ver um exemplo de utilização. Meus filhos batizaram este efeito de "ESCALADA".

PROGRAMA 26

(36 bytes)

Ø REM 36 caracteres quaisquer

INÍCIO	16514	3E	97		LD A,151
	16516	2A	0C	40	LD HL,(16396)
	16519	11	D5	02	LD DE,725
	16522	19			ADD HL,DE
	16523	06	16		LD B,22
	16525	11	21	00	LD DE,33
	16528	77			LD (HL),A
	16529	CD	9E	40	CALL 16542 (CALL PAUSA)
	16532	2B			DEC HL
	16533	77			LD (HL),A
IMPRIME	16534	CD	9E	40	CALL 16542 (CALL PAUSA)
	16537	ED	52		SBC HL,DE
	16539	10	F3		DJNZ -13 (DJNZ IMPRIME)
	16541	C9			RET
	16542	F5			PUSH AF
PAUSA	16543	3E	00		LD A,Ø

16545	3D	
16546	20	FD
16548	F1	
16549	C9	

DEC A
JR NZ,-3
POP AF
RET

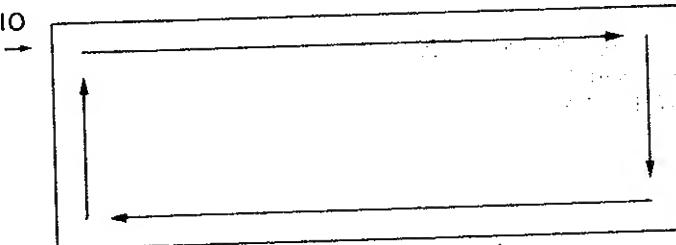
9.5

EXERCÍCIO

- 75 Este exercício, além de fornecer uma bonita rotina, que poderá ser usada em seus programas BASIC, é o que nos falta para o perfeito entendimento do CLS ESPIRAL.

Faça um programa (que tenha no final a sub-rotina de pausa) que coloque uma moldura na tela, mas desta maneira:

INÍCIO



A pausa torna o efeito visível.

Depois diminua a pausa, fazendo o POKE adequado, e descubra como o efeito fica mais bonito para seu gosto.

9.6

O CLS ESPIRAL

Bem, estamos chegando ao final. Divirta-se com este programa.

Ø REM 91 caracteres quaisquer

16514	3E	80	CD	93	40	3E	00	CD
16522	93	40	01	00	00	CD	F5	08
16530	C9	2A	0C	40	0E	15	06	20
16538	50	23	77	CD	D5	40	15	20
16546	F8	05	59	C5	01	21	00	09
16554	77	CD	D5	40	1D	20	F8	C1
16562	0D	50	2B	77	CD	D5	40	15
16570	20	F8	5F	78	FE	0B	C8	7B
16578	05	59	C5	01	21	00	ED	42
16586	77	CD	D5	40	1D	20	F7	C1
16594	0D	18	C5	F5	3E	00	3D	20
16602	FD	F1	C9					

Se você deseja usar apenas o efeito em seus programas BASIC, descubra a pausa mais adequada para seu gosto, faça POKE 16599,XX e substitua a instrução CLS por RAND USR 16514, e seu programa BASIC tem um interessante visual...

Para demonstração do efeito, há programas BASIC interessantes. Veja:

```
10 FOR F=0 TO 21
20 PRINT AT F,F;"COMPUTADOR"
30 NEXT F
40 RAND USR 16514
50 PRINT "TELA LIMPA"
```

Ou então este:

```
10 LIST
20 RAND USR 16514
30 PRINT "TELA LIMPA"
```

Este também:

```
10 FOR F=255 TO 0 STEP -17
20 POKE 16599,F
30 RAND USR 16514
40 NEXT F
```

9.7

EXERCÍCIOS

- 76 Faça a desmontagem completa do programa CLS ESPIRAL. Esforce-se por entender a finalidade de cada instrução — você aprenderá muito. Introduza rótulos; isto vai ajudar a sua compreensão. Siga o programa como se você fosse o computador e descubra o que tem que ser feito após cada instrução — e você entenderá... Esforce-se: *sem esforço nada se obtém.*
- 77 Que instrução BASIC deve ser dada em lugar de RAND USR 16514 para que o CLS ESPIRAL funcione limpando a tela *apenas* imprimindo os espaços? Estude a desmontagem que você fez, e a resposta é fácil.....

9.8

PALAVRAS FINAIS

Viemos juntos até aqui, e é hora de nos separarmos. Espero que você tenha aprendido muito. Mas também que tenha se divertido muito.

Anatole France, em *Le Crime de Sylvestre Bonnard* (1881) disse que “Não é divertindo-se que uma pessoa aprende”. Eu creio que é *apenas* divertindo-se que uma pessoa pode aprender.

É incrível o quanto fizemos com tão poucas instruções. Há tantas mais...

Talvez nos encontremos, eu, você e a linguagem de máquina, no volume 2 deste livro. Até lá!

RESPOSTAS DOS EXERCÍCIOS

72 Ø REM 30 caracteres quaisquer

RÓTULO	ENDEREÇO	CÓDIGOS HEXA			MNEMÔNICOS
INÍCIO SALTA NL PEEK	16514	2A	0C	40	LD HL,(16396)
	16517	0E	16		LD C,22
	16519	23			INC HL
	16520	06	20		LD B,32
	16522	7E			LD A,(HL)
	16523	C6	80		ADD A,128
	16525	77			LD (HL),A
	16526	CD	98	40	CALL 16536 (CALL PAUSA)
	16529	23			INC HL
	16530	10	F6		DJNZ -10 (DJNZ PEEK)
PAUSA	16532	0D			DEC C
	16533	20	F0		JR NZ,-16 (JR NZ,SALTA NL)
	16535	C9			RET
	16536	F5			PUSH AF
	16537	3E	00		LD A,0
	16539	3D			DEC A
	16540	20	FD		JR NZ,-3
	16542	F1			POP AF
	16543	C9			RET

73 Ø REM 40 caracteres quaisquer

RÓTULO	ENDEREÇO	CÓDIGOS HEXA			MNEMÔNICOS
INÍCIO SALTA NL POKE 1	16514	2A	0C	40	LD HL,(16396)
	16517	06	16		LD B,22
	16519	0E	16		LD C,22
	16521	23			INC HL
	16522	51			LD D,C
	16523	1E	20		LD E,32
	16525	36	17		LD (HL),23
	16527	CD	A4	40	CALL 16548 (CALL PAUSA)
	16530	23			INC HL
	16531	1D			DEC E
POKE 2	16532	15			DEC D
	16533	20	F6		JR NZ,-10 (JR NZ,POKE 1)
	16535	36	97		LD (HL),151
	16537	CD	A4	40	CALL 16548 (CALL PAUSA)
	16540	23			INC HL
	16541	1D			DEC E
	16542	20	F7		JR NZ,-9 (JR NZ,POKE 2)

	16544	0D		DEC C
PAUSA	16545	10	E6	DJNZ -26 (DJNZ SALTA NL)
	16547	C9		RET
	16548	3E	00	LD A,0
	16550	3D		DEC A
	16551	20	FD	JR NZ,-3
	16553	C9		RET

Duas observações:

- 1º) Como o acumulador não era usado no programa, não havia a necessidade de preservá-lo e depois recuperá-lo com PUSH AF e POP AF.
- 2º) Se desejar variar o tempo de pausa, digite (modo direto) POKE 16549,192 (por exemplo).

74 Vejamos o CLS HORIZONTAL

Ø REM 41 caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS
INÍCIO	16514	3E 80	LD A,128
	16516	CD 90 40	CALL (CALL ENCHE TELA)
	16519	3E 00	LD A,0
	16521	CD 90 40	CALL 16528 (CALL ENCHE TELA)
	16524	CD F5 08	CALL 2293 (CALL PRINT AT)
	16527	C9	RET
ENCHE TELA	16528	2A 0C 40	LD HL,(16396)
	16531	0E 16	LD C,22
SALTA NL	16533	23	INC HL
	16534	06 20	LD B,32
IMPRIME	16536	77	LD (HL),A
	16537	CD AE 40	CALL 16547 (CALL PAUSA)
	16540	23	INC HL
	16541	10 F9	DJNZ -7 (DJNZ IMPRIME)
	16543	0D	DEC C
	16544	20 F3	JR NZ,-13 (JRNZ, SALTA NL)
PAUSA	16546	C9	RET
	16547	F5	PUSH AF
	16548	3E 00	LD A,0
	16550	3D	DEC A
	16551	20 FD	JR NZ,-3
	16553	F1	POP AF
	16554	C9	RET

Para acelerar, digite por exemplo (modo direto) POKE 16549,128.

75 Ø REM 54 caracteres quaisquer

INÍCIO	16514	3E	97		LD A,151
	16516	2A	0C	40	LD HL,(16396)
	16519	11	21	00	LD DE,33
	16522	06	20		LD B,32
LOOP 1	16524	23			INC HL
	16525	77			LD (HL),A
	16526	CD	B0	40	CALL 16560 (CALL PAUSA)
	16529	10	F9		DJNZ -7 (DJNZ LOOP 1)
LOOP 2	16531	06	15		LD B,21
	16533	19			ADD HL,DE
	16534	77			LD (HL),A
	16535	CD	B0	40	CALL 16560 (CALL PAUSA)
	16538	10	F9		DJNZ -7 (DJNZ LOOP 1)
LOOP 3	16540	06	1F		LD B,31
	16542	2B			DEC HL
	16543	77			LD (HL),A
	16544	CD	B0	40	CALL 16560 (CALL PAUSA)
	16547	10	F9		DJNZ -7 (DJNZ LOOP 3)
LOOP 4	16549	06	14		LD B,20
	16551	ED	52		SBC HL,DE
	16553	77			LD (HL),A
	16554	CD	B0	40	CALL 16560 (CALL PAUSA)
	16557	10	F8		DJNZ -8
PAUSA	16559	C9			RET
	16560	F5			PUSH AF
	16561	3E	00		LD A,0
	16563	3D			DEC A
	16564	20	FD		JR NZ,-3
	16566	F1			POP AF
	16567	C9			RET

Para reduzir a pausa, digite por exemplo (modo direto) POKE 16562,128.

76 Ø REM 91 caracteres quaisquer

RÓTULOS	ENDEREÇOS	CÓDIGOS HEXA	MNEMÔNICOS		
INÍCIO	16514	3E	80	LD A,128	
	16516	CD	93	CALL 16531 (CALL TELA)	
	16519	3E	00	LD A,0	
	16521	CD	93	40	CALL 16531 (CALL TELA)
	16524	01	00	00	LD BC,0
	16527	CD	F5	08	CALL 2293 (CALL PRINT AT 0,0)
	16530	C9			RET
TEL A	16531	2A	0C	40	LD HL,(16396)
	16534	0E	15		LD C,21

COMEÇO LOOP 1	16536	06	20	LD B,32	
	16538	50		LD D,B	
	16539	23		INC HL	
	16540	77		LD (HL),A	
	16541	CD	D5	40	CALL 16597 (CALL PAUSA)
	16544	15		DEC D	
	16545	20	F8	JR NZ,-8 (JR NZ,LOOP 1)	
	16547	05		DEC B	
	16548	59		LD E,C	
	16549	C5		PUSH BC	
	16550	01	21	00	LD BC,33
	16553	09		ADD HL,BC	
	16554	77		LD (HL),A	
	16555	CD	D5	40	CALL 16597 (CALL PAUSA)
	16558	1D		DEC E	
	16559	20	F8	JR NZ,-8 (JR NZ,LOOP 2)	
	16561	C1		POP BC	
LOOP 2	16562	0D		DEC C	
	16563	50		LD D,B	
	16564	2B		DEC HL	
	16565	77		LD (HL),A	
	16566	CD	D5	40	CALL 16597 (CALL PAUSA)
	16569	15		DEC D	
	16570	20	F8	JR NZ,-8 (JR NZ,LOOP 3)	
	16572	5F		LD E,A	
	16573	78		LD A,B	
	16574	FE	0B	CP 11	
LOOP 3	16576	C8		RET Z	
	16577	7B		LD A,E	
	16578	05		DEC B	
	16579	59		LD E,C	
	16580	C5		PUSH BC	
	16581	01	21	00	LD BC,33
	16584	ED	42	SBC HL,BC	
	16586	77		LD (HL),A	
	16587	CD	D5	40	CALL 16597 (CALL PAUSA)
	16590	1D		DEC E	
LOOP 4	16591	20	F7	JR NZ,-9 (JR NZ,LOOP 4)	
	16593	C1		POP BC	
	16594	0D		DEC C	
	16595	18	C5	JR,-59 (JR COMEÇO)	
	16597	F5		PUSH AF	
	16598	3E	00	LD A,0	
	16600	3D		DEC A	
	16601	20	FD	JR NZ,-3	
	16603	F1		POP AF	
	16604	C9		RET	
PAUSA					

Uma observação sobre a instrução do endereço 16576. O RETURN (RET) ao qual estamos acostumados é o RETURN incondicional. Também há RETURN condicional, vejamos:

INSTRUÇÃO	SIGNIFICADO	CÓDIGO HEXA
RET Z	RETURN IF ZERO	C8
RET NZ	RETURN IF NOT ZERO	C0
RET C	RETURN IF CARRY	D8
RET NC	RETURN IF NOT CARRY	D0
RET PE	RETURN IF PARITY IS EVEN	E8
RET PO	RETURN IF PARITY IS ODD	E0
RET M	RETURN IF MINUS	F8
RET P	RETURN IF PLUS	F0

- 77 Basta fazer como linha de programa RAND USR 16519. (Como comando direto é inútil — embora funcione — pois a execução de comandos diretos é *precedida* de um CLS.) Em 16519 já entramos no programa em LD A,0.

Experimente:

```
1000 LIST  
1010 RAND USR 16519  
1020 GOTO 1000
```

O efeito de apagar a tela em espiral é interessante, e pode ser usado em programas BASIC, jogos etc...

A UCP de um microcomputador é constituída de um microprocessador.

O ZX81 da Sinclair, bem como todos os compatíveis brasileiros (TK82, 83 e 85; CP-200; RINGO; AS-1000), baseia-se no microprocessador Z-80, da Zilog.

Comercializado desde 1976, o Z-80 integra 8000 transistores e é uma ampliação do 8080 da Intel: aceita todas as instruções do 8080 e mais algumas.

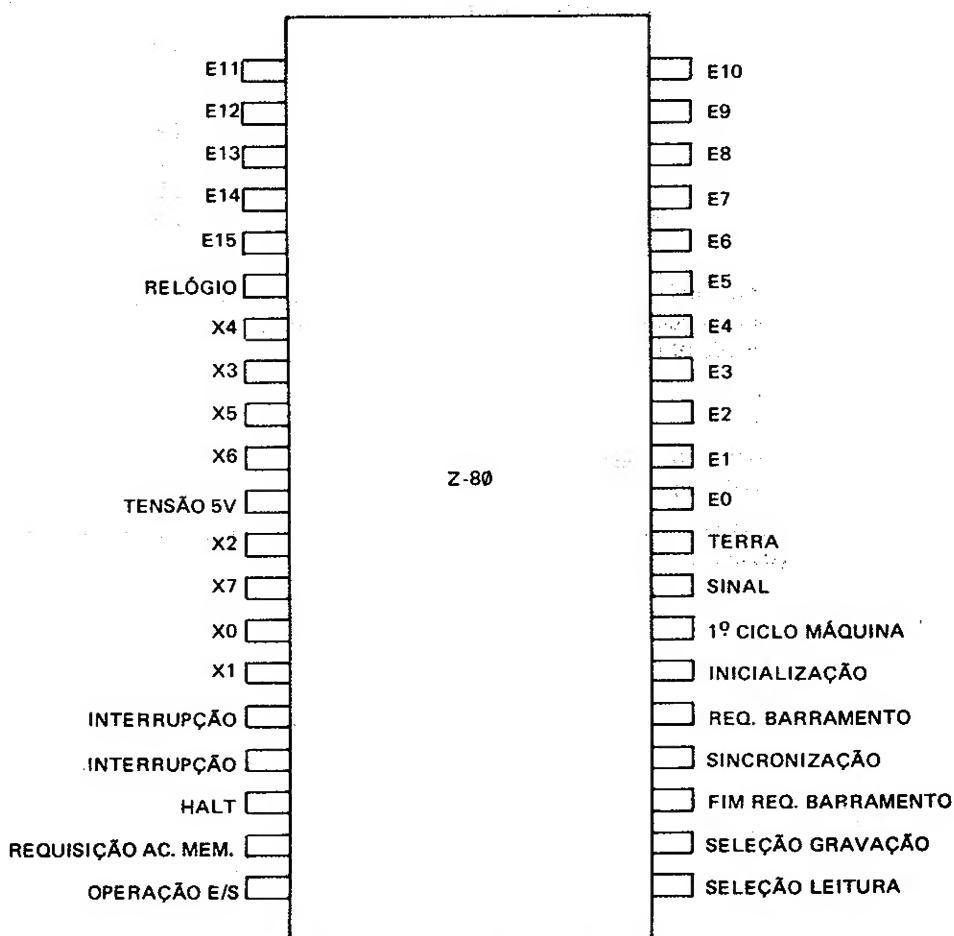


FIG. 1 — Esquema das conexões externas do microprocessador Z-80. O barramento de endereços está representado pela letra E (E0, E1,..., E15) e o barramento de dados pela letra X (X0, X1, ..., X7).

É fabricado com tecnologia N.MOS (MOS = metal oxide semiconductors); N = transistores de canal N). Alimenta-se com 5 volts. Integra uma pastilha com 40 conexões com o exterior, e pode gerir uma memória de até 64 kbytes.

A principal característica do Z-80, que o destaca dos demais, é o fato de dispor de vários registradores duplicados (alternativos), com os quais se consegue grande versatilidade. Entre os registradores duplicados, merecem destaque especial os conjuntos de registradores gerais B, C, D, E, H e L e seus alternativos B', C', D', E', H' e L', cada um de 8 bits, que podem ser usados aos pares, formando registradores de 16 bits: BC, DE, HL, BC', DE' e HL'.

Os 22 registradores internos do Z-80 são os seguintes:

- 2 registradores acumuladores, A e A', de 8 bits.
- 2 registradores de estado, F e F', de 8 bits.
- 12 registradores gerais: B e B', C e C', D e D', E e E', H e H', L e L', de 8 bits.
- 1 apontador da pilha, SP (stack pointer), de 16 bits.
- 1 contador de programa, PC (program counter), de 16 bits.
- 2 registradores indexados, IX e IY, de 16 bits.
- 1 registrador reavivador (refresh), R, de 8 bits.
- 1 registrador de interrupção, I, de 8 bits.

Algumas observações:

- 1^a) O acumulador A pode se juntar ao registrador de estado F, obtendo-se o registrador de 16 bits AF. A mesma coisa é válida para A' e F', obtendo-se AF'.
- 2^a) Os registradores A' e F' são usados pelo hardware dos Sinclair para operar no modo SLOW; assim sendo, não os use em SLOW ou... CRASH!
- 3^a) IX e IY são usados para facilitar alguns tipos de endereçamento; mas há algumas peculiaridades nos micros Sinclair:
 - IX é utilizado pelo hardware de SLOW, logo vale o mesmo que dissemos na segunda observação;
 - IY aponta para o endereço 4000h (16384d), ou seja, o endereço inicial das variáveis do sistema;
- 4^a) O registrador I (de interrupção) deve ter o valor 1Eh (30d) retornando ao BASIC.
- 5^a) Podemos imaginar conforme Fig. 2. os registradores internos do Z-80.
- 6^a) O registrador de estado (F) merece especial atenção. Seis de seus oito bits fornecem informações importantes sobre as condições internas do microprocessador. O esquema da Fig. 3 auxiliará nossa compreensão.

CONJUNTO PRINCIPAL:

A	[] [] [] [] [] [] []	F	[] [] [] [] [] [] []
B	[] [] [] [] [] [] []	C	[] [] [] [] [] [] []
D	[] [] [] [] [] [] []	E	[] [] [] [] [] [] []
H	[] [] [] [] [] [] []	L	[] [] [] [] [] [] []
SP	[] [] [] [] [] [] [] []		
PC	[] [] [] [] [] [] [] []		
IX	[] [] [] [] [] [] [] []		
IY	[] [] [] [] [] [] [] []		
I	[] [] [] [] [] [] [] []		
R	[] [] [] [] [] [] [] []		

CONJUNTO ALTERNATIVO:

A'	[] [] [] [] [] [] []	F'	[] [] [] [] [] [] []
B'	[] [] [] [] [] [] []	C'	[] [] [] [] [] [] []
D'	[] [] [] [] [] [] []	E'	[] [] [] [] [] [] []
H'	[] [] [] [] [] [] []	L'	[] [] [] [] [] [] []

FIG. 2 — O duplo conjunto de registradores internos é uma das características básicas do microprocessador Z-80.

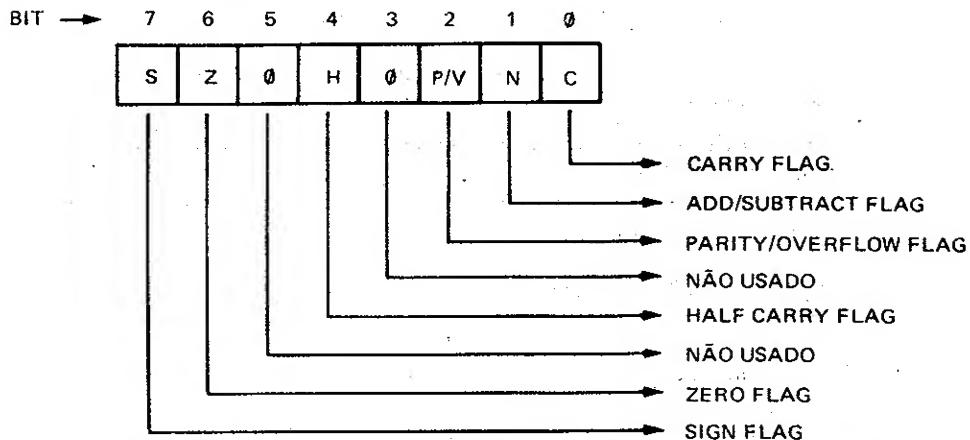


FIG. 3 — O registrador F do microprocessador Z-80 é bastante diferente dos registradores de estado de outros microprocessadores.

Vamos estudar cada um dos bits.

- BIT 0 (CARRY FLAG) – controla se houve ou não transporte (carry)
Houve transporte → 1
Não houve → 0
- BIT 1 (ADD/SUBTRACT FLAG) – controla se houve soma ou subtração
Houve subtração → 1
Houve soma → 0
- BIT 2 (PARITY/OVERFLOW FLAG) – em operações lógicas, controla a paridade, ou seja, se o número de 0 (ou 1) do conteúdo do registrador (expresso em binário) é par ou ímpar.
Paridade par → 1
Paridade ímpar → 0
 - em operações aritméticas, controla o transbordamento (overflow)
Houve transbordamento → 1
Não houve → 0
- BIT 3 – não é usado.
- BIT 4 (HALF CARRY FLAG) – controla se houve transbordamento ao somar a primeira metade do byte; esta flag é usada internamente pelo microprocessador, e não está à disposição (diretamente) do programador.
Houve transporte → 1
Não houve → 0
- BIT 5 – não é usado.
- BIT 6 (ZERO FLAG) – controla se o conteúdo do registrador é ou não zero; é útilíssimo verificar seu estado após operações aritméticas. *CUIDADO!* A flag Z não é afetada por operações com *pares* de registradores.
Conteúdo zero → 1
Conteúdo não zero → 0
- BIT 7 (SIGN FLAG) – controla o estado do sinal.
Número negativo → 1
Número positivo → 0

7º) O microprocessador Z-80 é, em nossa opinião, o melhor microprocessador de 8 bits disponível no mercado. Orgulhe-se de seu pequenino Sinclair – afinal de contas, ele tem um Z-80!

CÓDIGO DECIMAL	CÓDIGO HEXADECIMAL	CARACTERE
0	00	Espaço
1	01	
2	02	
3	03	
4	04	
5	05	
6	06	
7	07	
8	08	
9	09	
10	0A	
11	0B	
12	0C	
13	0D	
14	0E	
15	0F	
16	10	
17	11	
18	12	
19	13	
20	14	
21	15	
22	16	
23	17	
24	18	
25	19	
26	1A	
27	1B	
28	1C	
29	1D	
30	1E	
31	1F	
32	20	
33	21	
34	22	
35	23	
36	24	
37	25	
38	26	

CÓDIGO DECIMAL	CÓDIGO HEXADECIMAL	CARACTERE
39	27	B
40	28	C
41	29	D
42	2A	E
43	2B	F
44	2C	G
45	2D	H
46	2E	I
47	2F	J
48	30	K
49	31	L
50	32	M
51	33	N
52	34	O
53	35	P
54	36	Q
55	37	R
56	38	S
57	39	T
58	3A	U
59	3B	V
60	3C	W
61	3D	X
62	3E	Y
63	3F	Z
64	40	RND
65	41	INKEY\$
66	42	PI
128	80	
129	81	
130	82	
131	83	
132	84	
133	85	
134	86	
135	87	
136	88	
137	89	
138	8A	
139	8B	"
140	8C	£
141	8D	\$
142	8E	:
143	8F	?
144	90	(

CÓDIGO DECIMAL	CÓDIGO HEXADECIMAL	CARACTERE
145	91)
146	92	>
147	93	<
148	94	=
149	95	+
150	96	-
151	97	*
152	98	/
153	99	;
154	9A	,
155	9B	.
156	9C	0
157	9D	1
158	9E	2
159	9F	3
160	A0	4
161	A1	5
162	A2	6
163	A3	7
164	A4	8
165	A5	9
166	A6	A
167	A7	B
168	A8	C
169	A9	D
170	AA	E
171	AB	F
172	AC	G
173	AD	H
174	AE	I
175	AF	J
176	B0	K
177	B1	L
178	B2	M
179	B3	N
180	B4	O
181	B5	P
182	B6	Q
183	B7	R
184	B8	S
185	B9	T
186	BA	U
187	BB	V
188	BC	W
189	BD	X

CÓDIGO DECIMAL	CÓDIGO HEXADECIMAL	CARACTERE
190	BE	Y
191	BF	Z
192	C0	" "
193	C1	AT
194	C2	TAB
196	C4	CODE
197	C5	VAL
198	C6	LEN
199	C7	SIN
200	C8	COS
201	C9	TAN
202	CA	ASN
203	CB	ACS
204	CC	ATN
205	CD	LN
206	CE	EXP
207	CF	INT
208	D0	SOR
209	D1	SGN
210	D2	ABS
211	D3	PEEK
212	D4	USR
213	D5	STR\$
214	D6	CHR\$
215	D7	NOT
216	D8	**
217	D9	OR
218	DA	AND
219	DB	<=
220	DC	>=
221	DD	<>
222	DE	THEN
223	DF	TO
224	E0	STEP
225	E1	LPRINT
226	E2	LLIST
227	E3	STOP
228	E4	SLOW
229	E5	FAST
230	E6	NEW
231	E7	SCROLL
232	E8	CONT
233	E9	DIM
234	EA	REM
235	EB	FOR

CÓDIGO DECIMAL	CÓDIGO HEXADECIMAL	CARACTERE
236	EC	GOTO
237	ED	GOSUB
238	EE	INPUT
239	EF	LOAD
240	F0	LIST
241	F1	LET
242	F2	PAUSE
243	F3	NEXT
244	F4	POKE
245	F5	PRINT
246	F6	PLOT
247	F7	RUN
248	F8	SAVE
249	F9	RAND
250	FA	IF
251	FB	CLS
252	FC	UNPLOT
253	FD	CLEAR
254	FE	RETURN
255	FF	COPY

A Sinclair usa ainda os seguintes "caracteres"; eles (obviamente) não são imprimíveis e aparecem "impressos" como ?. Todos os códigos não-usados (de 67 a 111, de 122 a 125, e ainda 195) são também "impressos" como ?.

112	70	cursor para cima
113	71	cursor para baixo
114	72	cursor para esquerda
115	73	cursor para direita
116	74	GRAPHICS
117	75	EDIT
118	76	NEW LINE (ENTER)
119	77	RUBOUT (DELETE)
120	78	MODO
121	79	FUNCTION
126	7E	NÚMERO
127	7F	CURSOR

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Exemplos de utilização: 201 = C9

118 = 76

58 = 3A

Apêndice 3

TABELA DE CONVERSÃO DECIMAL/HEXADECIMAL

(NÚMEROS NEGATIVOS DE -1 A -128)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Exemplos de utilização: $-34 = DE$

$-118 = 8A$

$-5 = FB$

ENDEREÇO		Nº DE BYTES	NOME SINCLAIR	FUNÇÃO
DECIMAL	HEXA			
16384	4000	1	ERR.NR	Código de denotação –1
16385	4001	1	FLAGS	Várias flags para controlar o sistema BASIC
16386	4002	2	ERR.SP	Ponteiro da pilha após uma GOSUB
16388	4004	2	RAMTOP	Endereço do topo da RAM existente ou alterada pelo usuário
16390	4006	1	MODE	Especifica o cursor que está sendo usado na linha atual
16391	4007	2	PPC	Número da linha sendo executada
16393	4009	1	VERSN	Ponto inicial da RAM a ser gravado a partir de SAVE
16394	400A	2	E.PPC	Número da linha onde está o cursor
16396	400C	2	D.FILE	Ponteiro do arquivo de imagem
16398	400E	2	DF.CC	Endereço da posição de PRINT no arquivo de imagem
16400	4010	2	VARS	Ponteiro da área das variáveis
16402	4012	2	DEST	Endereço da variável em atribuição
16404	4014	2	E.LINE	Ponteiro para área de trabalho
16406	4016	2	CH.ADD	Endereço do próximo caractere a ser interpretado
16408	4018	2	X.PTR	Endereço do caractere que precede o cursor de erro de sintaxe
16410	401A	2	STKBOT	Ponteiro do início da pilha do calculador
16412	401C	2	STKEND	Ponteiro do final da pilha do calculador
16414	401E	1	BERG	Usado pelo calculador em ponto flutuante
16415	401F	2	MEM	Ponteiro da área da memória onde os cálculos são feitos
16417	4021	1		Não é usado
16418	4022	1	DFSZ	Número de linhas na parte inferior da tela
16419	4023	2	S.TOP	Número da linha superior da tela ao listar
16421	4025	2	LAST.K	Valor da última tecla pressionada
16423	4027	1	DB.ST	Estado de debounce do teclado
16424	4028	1	MARGIN	Número de linhas em branco acima ou abaixo da imagem

ENDEREÇO		Nº DE BYTES	NOME SINCLAIR	FUNÇÃO
DECIMAL	HEXA			
16425	4029	2	NXTLIN	Endereço da próxima linha do programa a ser executada
16427	402B	2	OLDPPC	Número da linha para a qual CONT salta
16429	402D	1	FLAGX	Várias flags
16430	402E	2	STRLEN	Informações sobre atribuição de strings
16432	4030	2	T.ADDR	Ponteiro para tabela de syntaxe
16434	4032	2	SEED	"Semente" para o gerador de números randômicos
16436	4034	2	FRAMES	Contador de quadros apresentados na tela
16438	4036	2	COORDS	Coordenadas do último ponto plotado
16440	4038	1	PR.CC	Byte menos significativo da posição de LPRINT; o byte mais significativo é 40h
16441	4039	2	S.POSN	Número da coluna e da linha, respectivamente, para a posição de PRINT
16443	403B	1	CDFLAG	Flags relativas a SLOW e FAST
16444	403C	33	PRBUFF	Buffer da impressora, 33º caractere é um NEW LINE
16477	405D	30	MEMBOT	Área que pode ser usada como memória do calculador
16507	407B	2		Não são usados

SINCLAIR	CP-200	TK85
ERR.NR	ERRO-1	CODR
FLAGS	FLAGS	BAND
ERR.SP	RETGSB	ENSP
RAMTOP	MEMTOP	RTP
MODE	MODO	MODO
PPC		CPB
VERSN	VERSÃO	VERSN
E.PPC	NUMLI	LPC
D.FILE	MAPTELA	DFILE
DF.CC		POSPR
VARS	VARS	VARS
DEST	DEST	DEST
E.LINE	LIDIGIT	ELINE
CH.ADD	PROX-CAR	ENCAR
X.PTR	XPTR	ENSX
STKBOT	STKCOM	PILFUN
STKEND	STKFIM	PILFIM
BERG	REGIB	CALREG
MEM	MEM	MEM
não-usado	não-usado	não-usado
DF.SZ	DF-SZ	DFSZ
S.TOP	PROGTOP	LTOP
LAST.K	ULTIMAT	ULTK
DB.ST		
MARGIN	MARGEN	HARG
NXTLIN	PRXLIN	PXLN
OLDPPC	LINCONT	VCPB
FLAGX	FLAGX	BANDX
STRLEN	COMSTR	LENCA
T.ADDR	ENDSTX	SXEN
SEED	SEED	SEMT
FRAMES	FRAMES	QUAD
COORDS	COORDX	CORDX
	COORDY	CORDY
PR.CC	LPBYTE	PR-CC
S.POSN	COLUNA	COLPR
	LINHA	LINPR
CDFLAG	FLAGCD	BANCO
PRBUFF	BUF IMP	PRBUFF
MEMBOT	MEMBOT	MEMBO
não-usado	não-usado	não-usado

Você pode usar as variáveis do sistema para obter informações úteis. Vamos ver neste apêndice algumas possibilidades.

Mas... o importante não é apenas saber que tal "fórmula", tipo PRINT PEEK não-sai-lá-o-que, faz tal ou qual coisa, mas sim entender o que se está fazendo.

Para isto é necessário entender a organização da RAM do nosso micro. A figura abaixo nos ajudará.

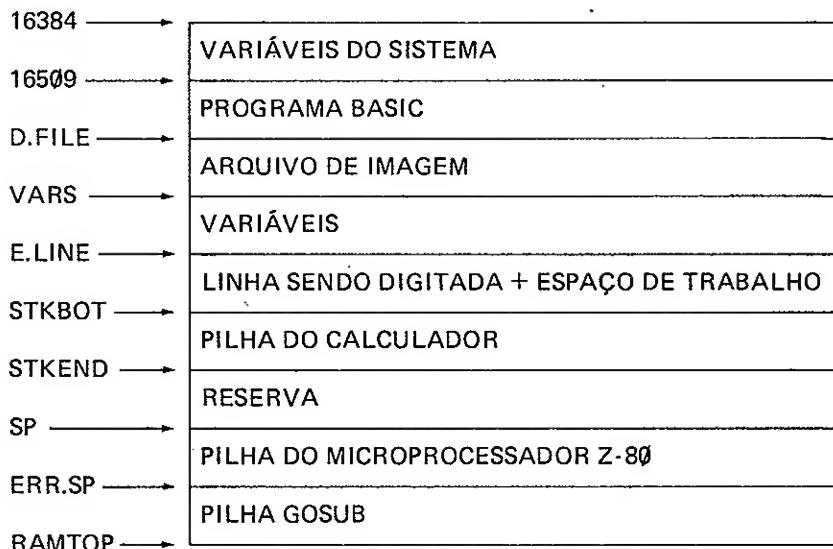


FIG. 1 — A organização da armazenagem na memória RAM.

Como você pode perceber, só há dois endereços fixos: o de início das variáveis do sistema, que é o próprio início da RAM, 16384 (4000h); e o do início do programa BASIC, 16509 (407D). Todos os outros endereços, até o final, são variáveis. Eles são apontados pelas variáveis do sistema indicadas na figura.

Para obter o endereço apontado pela variável do sistema, basta fazer o seguinte:

- procurar o endereço *decimal* da variável do sistema na tabela do apêndice 5.
exemplo: D.FILE..... 16396
- chamando a este endereço de E, o endereço apontado pela variável do sistema é dado por
 $\text{PEEK } E + 256 * \text{PEEK } (E+1)$

exemplo: logo, para obter o endereço apontado por D.FILE, basta fazer

PRINT PEEK 16396+256*PEEK 16397

Vamos então às *sub-rotinas* de serviço.

1º) CAPACIDADE DE MEMÓRIA RAM INSTALADA

Eis aí uma coisa útil de se verificar, principalmente quando se usa expansão de memória.

Observando a Fig. 1, vemos que a RAM vai do endereço 16384 até o endereço apontado por RAMTOP.

Logo, para obter o número de bytes de memória RAM instalada, basta fazer

PRINT PEEK 16388+256*PEEK 16389-16384

Digite isto como um comando direto e NEW LINE.

Se você dividir o número obtido por 1024, obterá quantos K de memória tem o seu micro, pois o 1 Kbyte = 1024 bytes. Esta observação também é válida para as outras sub-rotinas de serviço deste apêndice.

2º) NÚMERO DE BYTES OCUPADOS PELO PROGRAMA BASIC

Vimos na Fig. 1 que o programa BASIC vai do endereço 16509 até o endereço apontado por D.FILE.

Logo, para obter o número de bytes ocupados pelo seu programa BASIC, faça

PRINT PEEK 16396+256*PEEK 16397-16509

3º) NÚMERO TOTAL DE BYTES OCUPADOS PELO PROGRAMA BASIC

Seu programa BASIC gera um arquivo de imagem, e utiliza variáveis. Assim, ele na verdade vai de 16509 até o endereço apontado por E.LINE (observe a Fig. 1).

Logo, para obter o número de bytes ocupado pelo programa, incluindo as variáveis e o arquivo de imagem, faça

PRINT PEEK 16404+256*PEEK 16405-16509

4º) NÚMERO DE BYTES DE MEMÓRIA DISPONÍVEL

Se você for digitar um programa BASIC particularmente extenso, isto também é utilíssimo de se saber.

A memória RAM, não ocupada pelo seu programa BASIC, vai do endereço apontado por E.LINE até o endereço apontado por RAMTOP.

Assim, obtenha o número de bytes "livres" fazendo

PRINT (PEEK 16388+256*PEEK 16389) – (PEEK 16404+256*PEEK 16405)

Uma observação final: qualquer uma das quatro sub-rotinas apresentadas pode ser transformada em um pequeno programa Assembly. Como exemplo, aqui está a primeira delas (quantos K de memória).

Ø REM 13 caracteres quaisquer

16514	11	
16515	00	}
16516	40	
16517	2A	
16518	04	}
16519	40	
16520	C6	
16521	00	}
16522	ED	
16523	52	}
16524	44	LD B,H
16525	4D	LD C,L
16526	C9	RET

Para rodar o programa, digite (modo direto):

PRINT (USR 16514)/1024;"__K".

(Estas palavras são usadas como INSTRUÇÕES, logo não podem ser usadas como RÓTULOS.)

ADC	EX	NEG	RLCA
ADD	EXX	NOP	RLD
AND	HALT	OR	RR
BIT	IM	OTDR	RRA
CALL	IN	OTIR	RRC
CCF	INC	OUT	RRCA
CP	IND	OUTD	RRD
CPD	INDR	OUTI	RST
CPDR	INI	POP	SBC
CPI	INIR	PUSH	SCF
CPIR	JP	RES	SET
CPL	JR	RET	SLA
DAA	LD	RETI	SRA
DEC	LDD	RETN	SRL
DI	LDI	RL	SUB
DJNZ	LDIR	RLA	XOR
EI		RLC	

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
00	NOP	28	JR Z,DIS
01	LD BC,NN	29	ADD HL,HL
02	LD (BC),A	2A	LD HL,(NN)
03	INC BC	2B	DEC HL
04	INC B	2C	INC L
05	DEC B	2D	DEC L
06	LD B,N	2E	LD L,N
07	RLCA	2F	CPL
08	EX AF,AF'	30	JR NC,DIS
09	ADD HL,BC	31	LD SP,NN
0A	LD A,(BC)	32	LD (NN),A
0B	DEC BC	33	INC SP
0C	INC C	34	INC (HL)
0D	DEC C	35	DEC (HL)
0E	LD C,N	36	LD (HL),N
0F	RRCA	37	SCF
10	DJNZ DIS	38	JR C,DIS
11	LD DE,NN	39	ADD HL,SP
12	LD (DE),A	3A	LD A,(NN)
13	INC DE	3B	DEC SP
14	INC D	3C	INC A
15	DEC D	3D	DEC A
16	LD D,N	3E	LD A,N
17	RLA	3F	CCF
18	JR DIS	40	LD B,B
19	ADD HL,DE	41	LD B,C
1A	LD A,(DE)	42	LD B,D
1B	DEC DE	43	LD B,E
1C	INC E	44	LD B,H
1D	DEC E	45	LD B,L
1E	LD E,N	46	LD B,(HL)
1F	RRA	47	LD B,A
20	JR NZ,DIS	48	LD C,B
21	LD HL,NN	49	LD C,C
22	LD (NN),HL	4A	LD C,D
23	INC HL	4B	LD C,E
24	INC H	4C	LD C,H
25	DEC H	4D	LD C,L
26	LD H,N	4E	LD C,(HL)
27	DAA	4F	LD C,A

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
50	LD D,B	7E	LD A,(HL)
51	LD D,C	7F	LD A,A
52	LD D,D	80	ADD A,B
53	LD D,E	81	ADD A,C
54	LD D,H	82	ADD A,D
55	LD D,L	83	ADD A,E
56	LD D,(HL)	84	ADD A,H
57	LD D,A	85	ADD A,L
58	LD E,B	86	ADD A,(HL)
59	LD E,C	87	ADD A,A
5A	LD E,D	88	ADC A,B
5B	LD E,E	89	ADC A,C
5C	LD E,H	8A	ADC A,D
5D	LD E,L	8B	ADC A,E
5E	LD E,(HL)	8C	ADC A,H
5F	LD E,A	8D	ADC A,L
60	LD H,B	8E	ADC A,(HL)
61	LD H,C	8F	ADC A,A
62	LD H,D	90	SUB B
63	LD H,E	91	SUB C
64	LD H,H	92	SUB D
65	LD H,L	93	SUB E
66	LD H,(HL)	94	SUB H
67	LD H,A	95	SUB L
68	LD L,B	96	SUB (HL)
69	LD L,C	97	SUB A
6A	LD L,D	98	SBC A,B
6B	LD L,E	99	SBC A,C
6C	LD L,H	9A	SBC A,E
6D	LD L,L	9B	SBC A,H
6E	LD L,(HL)	9C	SBC A,L
6F	LD L,A	9D	SBC A,(HL)
70	LD (HL),B	9E	SBC A,A
71	LD (HL),C	9F	AND B
72	LD (HL),D	A0	AND C
73	LD (HL),E	A1	AND D
74	LD (HL),H	A2	AND E
75	LD (HL),L	A3	AND H
76	HALT	A4	AND L
77	LD (HL),A	A5	AND (HL)
78	LD A,B	A6	AND A
79	LD A,C	A7	XOR B
7A	LD A,D	A8	XOR C
7B	LD A,E	A9	XOR D
7C	LD A,H	AA	XOR E
7D	LD A,L	AB	

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
AC	XOR H	CB0F	RRC A
AD	XOR L	CB10	RL B
AE	XOR (HL)	CB11	RL C
AF	XOR A	CB12	RL D
B0	OR B	CB13	RL E
B1	OR C	CB14	RL H
B2	OR D	CB15	RL L
B3	OR E	CB16	RL (HL)
B4	OR H	CB17	RL A
B5	OR L	CB18	RR B
B6	OR (HL)	CB19	RR C
B7	OR A	CB1A	RR D
B8	CP B	CB1B	RR E
B9	CP C	CB1C	RR H
BA	CP D	CB1D	RR L
BB	CP E	CB1E	RR (HL)
BC	CP H	CB1F	RR A
BD	CP L	CB20	SLA B
BE	CP (HL)	CB21	SLA C
BF	CP A	CB22	SLA D
C0	RET NZ	CB23	SLA E
C1	POP BC	CB24	SLA H
C2	JP NZ,NN	CB25	SLA L
C3	JP NN	CB26	SLA (HL)
C4	CALL NZ,NN	CB27	SLA A
C5	PUSH BC	CB28	SRA B
C6	ADD A,N	CB29	SRA C
C7	RST 0	CB2A	SRA D
C8	RET Z	CB2B	SRA E
C9	RET	CB2C	SRA H
CA	JP Z,NN	CB2D	SRA L
CB00	RLC B	CB2E	SRA (HL)
CB01	RLC C	CB2F	SRA A
CB02	RLC D	CB38	SRL B
CB03	RLC E	CB39	SRL C
CB04	RLC H	CB3A	SRL D
CB05	RLC L	CB3B	SRL E
CB06	RLC (HL)	CB3C	SRL H
CB07	RLC A	CB3D	SRL L
CB08	RRC B	CB3E	SRL (HL)
CB09	RRC C	CB3F	SRL A
CB0A	RRC D	CB40	BIT 0,B
CB0B	RRC E	CB41	BIT 0,C
CB0C	RRC H	CB42	BIT 0,D
CB0D	RRC L	CB43	BIT 0,E
CB0E	RRC (HL)	CB44	BIT 0,H

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
CB45	BIT 0,L	CB73	BIT 6,E
CB46	BIT 0,(HL)	CB74	BIT 6,H
CB47	BIT 0,A	CB75	BIT 6,L
CB48	BIT 1,B	CB76	BIT 6,(HL)
CB49	BIT 1,C	CB77	BIT 6,A
CB4A	BIT 1,D	CB78	BIT 7,B
CB4B	BIT 1,E	CB79	BIT 7,C
CB4C	BIT 1,H	CB7A	BIT 7,D
CB4D	BIT 1,L	CB7B	BIT 7,E
CB4E	BIT 1,(HL)	CB7C	BIT 7,H
CB4F	BIT 1,A	CB7D	BIT 7,L
CB50	BIT 2,B	CB7E	BIT 7,(HL)
CB51	BIT 2,C	CB7F	BIT 7,A
CB52	BIT 2,D	CB80	RES 0,B
CB53	BIT 2,E	CB81	RES 0,C
CB54	BIT 2,H	CB82	RES 0,D
CB55	BIT 2,L	CB83	RES 0,E
CB56	BIT 2,(HL)	CB84	RES 0,H
CB57	BIT 2,A	CB85	RES 0,L
CB58	BIT 3,B	CB86	RES 0,(HL)
CB59	BIT 3,C	CB87	RES 0,A
CB5A	BIT 3,D	CB88	RES 1,B
CB5B	BIT 3,E	CB89	RES 1,C
CB5C	BIT 3,H	CB8A	RES 1,D
CB5D	BIT 3,L	CB8B	RES 1,E
CB5E	BIT 3,(HL)	CB8C	RES 1,H
CB5F	BIT 3,A	CB8D	RES 1,L
CB60	BIT 4,B	CB8E	RES 1,(HL)
CB61	BIT 4,C	CB8F	RES 1,A
CB62	BIT 4,D	CB90	RES 2,B
CB63	BIT 4,E	CB91	RES 2,C
CB64	BIT 4,H	CB92	RES 2,D
CB65	BIT 4,L	CB93	RES 2,E
CB66	BIT 4,(HL)	CB94	RES 2,H
CB67	BIT 4,A	CB95	RES 2,L
CB68	BIT 5,B	CB96	RES 2,(HL)
CB69	BIT 5,C	CB97	RES 2,A
CB6A	BIT 5,D	CB98	RES 3,B
CB6B	BIT 5,E	CB99	RES 3,C
CB6C	BIT 5,H	CB9A	RES 3,D
CB6D	BIT 5,L	CB9B	RES 3,E
CB6E	BIT 5,(HL)	CB9C	RES 3,H
CB6F	BIT 5,A	CB9D	RES 3,L
CB70	BIT 6,B	CB9E	RES 3,(HL)
CB71	BIT 6,C	CB9F	RES 3,A
CB72	BIT 6,D	CBA0	RES 4,B

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
CBA1	RES 4,C	CBCF	SET 1,A
CBA2	RES 4,D	CBD0	SET 2,B
CBA3	RES 4,E	CBD1	SET 2,C
CBA4	RES 4,H	CBD2	SET 2,D
CBA5	RES 4,L	CBD3	SET 2,E
CBA6	RES 4,(HL)	CBD4	SET 2,H
CBA7	RES 4,A	CBD5	SET 2,L
CBA8	RES 5,B	CBD6	SET 2,(HL)
CBA9	RES 5,C	CBD7	SET 2,A
CBAA	RES 5,D	CBD8	SET 3,B
CBAB	RES 5,E	CBD9	SET 3,C
CBAC	RES 5,H	CBDA	SET 3,D
CBAD	RES 5,L	CBDB	SET 3,E
CBAE	RES 5,(HL)	CBDC	SET 3,H
CBAF	RES 5,A	CBDD	SET 3,L
CBBØ	RES 6,B	CBDE	SET 3,(HL)
CBB1	RES 6,C	CBDF	SET 3,A
CBB2	RES 6,D	CBEØ	SET 4,B
CBB3	RES 6,E	CBE1	SET 4,C
CBB4	RES 6,H	CBE2	SET 4,D
CBB5	RES 6,L	CBE3	SET 4,E
CBB6	RES 6,(HL)	CBE4	SET 4,H
CBB7	RES 6,A	CBE5	SET 4,L
CBB8	RES 7,B	CBE6	SET 4,(HL)
CBB9	RES 7,C	CBE7	SET 4,A
CBBA	RES 7,D	CBE8	SET 5,B
CBBB	RES 7,E	CBE9	SET 5,C
CBBC	RES 7,H	CBEA	SET 5,D
CBBD	RES 7,L	CBEB	SET 5,E
CBBE	RES 7,(HL)	CBEC	SET 5,H
CBBF	RES 7,A	CBED	SET 5,L
CBCØ	SET Ø,B	CBEE	SET 5,(HL)
CBC1	SET Ø,C	CBEF	SET 5,A
CBC2	SET Ø,D	CBFØ	SET 6,B
CBC3	SET Ø,E	CBF1	SET 6,C
CBC4	SET Ø,H	CBF2	SET 6,D
CBC5	SET Ø,L	CBF3	SET 6,E
CBC6	SET Ø,(HL)	CBF4	SET 6,H
CBC7	SET Ø,A	CBF5	SET 6,L
CBC8	SET 1,B	CBF6	SET 6,(HL)
CBC9	SET 1,C	CBF7	SET 6,A
CBCA	SET 1,D	CBF8	SET 7,B
CBCB	SET 1,E	CBF9	SET 7,C
CBCC	SET 1,H	CBFA	SET 7,D
CBCD	SET 1,L	CBFB	SET 7,E
CBCE	SET 1,(HL)	CBFC	SET 7,H

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
CBFD	SET 7,L	DD86	ADD A,(IX+IND)
CBFE	SET 7,(HL)	DD8E	ADC A,(IX+IND)
CBFF	SET 7,A	DD96	SUB (IX+IND)
CC	CALL Z,NN	DD9E	SBC A,(IX+IND)
CD	CALL NN	DDA6	AND (IX+IND)
CE	ADC A,N	DDA8	XOR (IX+IND)
CF	RST 8	DDB6	OR (IX+IND)
DØ	RET NC	DDBE	CP (IX+IND)
D1	POP DE	DDCB..06	RLC (IX+IND)
D2	JP NC,NN	DDCB..0E	RRC (IX+IND)
D3	OUT N,A	DDCB..16	RL (IX+IND)
D4	CALL NC,NN	DDCB..1E	RR (IX+IND)
D5	PUSH DE	DDCB..26	SLA (IX+IND)
D6	SUB N	DDCB..2E	SRA (IX+IND)
D7	RST 10h	DDCB..3E	SRL (IX+IND)
D8	RET C	DDCB..46	BIT 0,(IX+IND)
D9	EXX	DDCB..4E	BIT 1,(IX+IND)
DA	JP C,NN	DDCB..56	BIT 2,(IX+IND)
DB	IN A,(N)	DDCB..5E	BIT 3,(IX+IND)
DC	CALL C,NN	DDCB..66	BIT 4,(IX+IND)
DD09	ADD IX,BC	DDCB..6E	BIT 5,(IX+IND)
DD19	ADD IX,DE	DDCB..76	BIT 6,(IX+IND)
DD21	LD IX,NN	DDCB..7E	BIT 7,(IX+IND)
DD22	LD (NN),IX	DDCB..86	RES Ø,(IX+IND)
DD23	INC IX	DDCB..8E	RES 1,(IX+IND)
DD29	ADD IX,IX	DDCB..96	RES 2,(IX+IND)
DD2A	LD IX,(NN)	DDCB..9F	RES 3,(IX+IND)
DD2B	DEC IX	DDCB..A6	RES 4,(IX+IND)
DD34	INC (IX+IND)	DDCB..AE	RES 5,(IX+IND)
DD35	DEC (IX+IND)	DDCB..B6	RES 6,(IX+IND)
DD36	LD (IX+IND),N	DDCB..BE	RES 7,(IX+IND)
DD39	ADD IX,SP	DDCB..C6	SET Ø,(IX+IND)
DD46	LD B,(IX+IND)	DDCB..CE	SET 1,(IX+IND)
DD4E	LD C,(IX+IND)	DDCB..D6	SET 2,(IX+IND)
DD56	LD D,(IX+IND)	DDCB..DE	SET 3,(IX+IND)
DD5E	LD E,(IX+IND)	DDCB..E6	SET 4,(IX+IND)
DD66	LD H,(IX+IND)	DDCB..EE	SET 5,(IX+IND)
DD6E	LD L,(IX+IND)	DDCB..F6	SET 6,(IX+IND)
DD70	LD (IX+IND),B	DDCB..FE	SET 7,(IX+IND)
DD71	LD (IX+IND),C	DDE1	POP IX
DD72	LD (IX+IND),D	DDE3	EX (SP),IX
DD73	LD (IX+IND),E	DDE5	PUSH IX
DD74	LD (IX+IND),H	DDE9	JP (IX)
DD75	LD (IX+IND),L	DDEB	EX DE,IX
DD77	LD (IX+IND),A	DDF9	LD SP,IX
DD7E	LD A,(IX+IND)	DE	SBC A,N

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
DF	RST 18h	ED69	OUT (C),L
EØ	RET PO	ED6A	ADC HL,HL
E1	POP HL	ED6B	LD HL,(NN)
E2	JP PO,NN	ED6F	RLD
E3	EX (SP),HL	ED72	SBC HL,SP
E4	CALL PO,NN	ED73	LD (NN),SP
E5	PUSH HL	ED78	IN A,(C)
E6	AND N	ED79	OUT (C),A
E7	RST 2Øh	ED7A	ADC HL,SP
E8	RET PE	ED7B	LD SP,(NN)
E9	JP (HL)	EDAØ	LDI
EA	JP PE,NN	EDA1	CPI
EB	EX DE,HL	EDA2	INI
EC	CALL PE,NN	EDA3	OUTI
ED4Ø	IN B,(C)	EDA8	LDD
ED41	OUT (C),B	EDA9	CPD
ED42	SBC HL,BC	EDAA	IND
ED43	LD (NN),BC	EDAB	OUTD
ED44	NEG	EDBØ	LDIR
ED45	RETN	EDB1	CPIR
ED46	IM Ø	EDB2	INIR
ED47	LD I,A	EDB3	OTIR
ED48	IN C,(C)	EDB8	LDDR
ED49	OUT (C),C	EDB9	CPDR
ED4A	ADC HL,BC	EDBA	INDR
ED4B	LD BC,(NN)	EDBB	OTDR
ED4D	RETI	EE	XOR N
ED4F	LD R,A	EF	RST 28h
ED5Ø	IN D,(C)	FØ	RET P
ED51	OUT (C),D	F1	POP AF
ED52	SBC HL,DE	F2	JP P,NN
ED53	LD (NN),DE	F3	DI
ED56	IM 1	F4	CALL P,NN
ED57	LD A,I	F5	PUSH AF
ED58	IN E,(C)	F6	OR N
ED59	OUT (C),E	F7	RST 3Øh
ED5A	ADC HL,DE	F8	RET M
ED5B	LD DE,(NN)	F9	LD SP,HL
ED5E	IM 2	FA	JP M,NN
ED5F	LD A,R	FB	EI
ED6Ø	IN H,(C)	FC	CALL M,NN
ED61	OUT (C),H	FDØ9	ADD IY,BC
ED62	SBC HL,HL	FD19	ADD IY,DE
ED63	LD (NN),HL	FD21	LD IY,NN
ED67	RRD	FD22	LD (NN),IY
ED68	IN L,(C)	FD23	INC IY

CÓDIGO DE MÁQUINA	COMANDO	CÓDIGO DE MÁQUINA	COMANDO
FD29	ADD IY,IY	FDCB..2E	SRA (IY+IND)
FD2A	LD IY,(NN)	FDCB..3E	SRL (IY+IND)
FD2B	DEC IY	FDCB..46	BIT 0,(IY+IND)
FD34	INC (IY+IND)	FDCB..4E	BIT 1,(IY+IND)
FD35	DEC (IY+IND)	FDCB..56	BIT 2,(IY+IND)
FD36	LD (IY+IND),N	FDCB..5E	BIT 3,(IY+IND)
FD39	ADD IY,SP	FDCB..66	BIT 4,(IY+IND)
FD46	LD B,(IY+IND)	FDCB..6E	BIT 5,(IY+IND)
FD4E	LD C,(IY+IND)	FDCB..76	BIT 6,(IY+IND)
FD56	LD D,(IY+IND)	FDCB..7E	BIT 7,(IY+IND)
FD5E	LD E,(IY+IND)	FDCB..86	RES 0,(IY+IND)
FD66	LD H,(IY+IND)	FDCB..8E	RES 1,(IY+IND)
FD6E	LD L,(IY+IND)	FDCB..96	RES 2,(IY+IND)
FD70	LD (IY+IND),B	FDCB..9E	RES 3,(IY+IND)
FD71	LD (IY+IND),C	FDCB..A6	RES 4,(IY+IND)
FD72	LD (IY+IND),D	FDCB..AE	RES 5,(IY+IND)
FD73	LD (IY+IND),E	FDCB..B6	RES 6,(IY+IND)
FD74	LD (IY+IND),H	FDCB..BE	RES 7,(IY+IND)
FD75	LD (IY+IND),L	FDCB..C6	SET 0,(IY+IND)
FD77	LD (IY+IND),A	FDCB..CE	SET 1,(IY+IND)
FD7E	LD A,(IY+IND)	FDCB..D6	SET 2,(IY+IND)
FD86	ADD A,(IY+IND)	FDCB..DE	SET 3,(IY+IND)
FD8E	ADC A,(IY+IND)	FDCB..E6	SET 4,(IY+IND)
FD96	SUB (IY+IND)	FDCB..EE	SET 5,(IY+IND)
FD9E	SBC A,(IY+IND)	FDCB..F6	SET 6,(IY+IND)
FDA6	AND (IY+IND)	FDCB..FE	SET 7,(IY+IND)
FDAE	XOR (IY+IND)	FDE1	POP IY
FDB6	OR (IY+IND)	FDE3	EX (SP),IY
FDBE	CP (IY+IND)	FDE5	PUSH IY
FDCB..06	RLC (IY+IND)	FDE9	JP (IY)
FDCB..0E	RRC (IY+IND)	FDEB	EX DE,IY
FDCB..16	RL (IY+IND)	FDF9	LD SP,IY
FDCB..1E	RR (IY+IND)	FE	CP N
FDCB..26	SLA (IY+IND)	FF	RST 38h

COMANDO	CÓDIGO	COMANDO	CÓDIGO
ADC A,(HL)	8E	AND A	A7
ADC A,(IX+IND)	DD8E	AND B	A0
ADC A,(IY+IND)	FD8E	AND C	A1
ADC A,A	8F	AND D	A2
ADC A,B	88	AND E	A3
ADC A,C	89	AND H	A4
ADC A,D	8A	AND L	A5
ADC A,E	8B	AND N	E6
ADC A,H	8C	BIT 0,(HL)	CB46
ADC A,L	8D	BIT 0,(IX+IND)	DDCB..46
ADC A,N	CE	BIT 0,(IY+IND)	FDCB..46
ADC HL,BC	ED4A	BIT 0,A	CB47
ADC HL,DE	ED5A	BIT 0,B	CB40
ADC HL,HL	ED6A	BIT 0,C	CB41
ADC HL,SP	ED7A	BIT 0,D	CB42
ADD A,(HL)	86	BIT 0,E	CB43
ADD A,(IX+IND)	DD86	BIT 0,H	CB44
ADD A,(IY+IND)	FD86	BIT 0,L	CB45
ADD A,A	87	BIT 1,(HL)	CB4E
ADD A,B	80	BIT 1,(IX+IND)	DDCB..4E
ADD A,C	81	BIT 1,(IY+IND)	FDCB..4E
ADD A,D	82	BIT 1,A	CB4F
ADD A,E	83	BIT 1,B	CB48
ADD A,H	84	BIT 1,C	CB49
ADD A,L	85	BIT 1,D	CB4A
ADD A,N	C6	BIT 1,E	CB4B
ADD HL,BC	09	BIT 1,H	CB4C
ADD HL,DE	19	BIT 1,L	CB4D
ADD HL,HL	29	BIT 2,(HL)	CB56
ADD HL,SP	39	BIT 2,(IX+IND)	DDCB..56
ADD IX,BC	DD09	BIT 2,(IY+IND)	FDCB..56
ADD IX,DE	DD19	BIT 2,A	CB57
ADD IX,IX	DD29	BIT 2,B	CB50
ADD IX,SP	DD39	BIT 2,C	CB51
ADD IY,BC	FD09	BIT 2,D	CB52
ADD IY,DE	FD19	BIT 2,E	CB53
ADD IY,IY	FD29	BIT 2,H	CB54
ADD IY,SP	FD39	BIT 2,L	CB55
AND (HL)	A6	BIT 3,(HL)	CB5E
AND (IX+IND)	DDA6	BIT 3,(IX+IND)	DDCB..5E
AND (IY+IND)	FDA6	BIT 3,(IY+IND)	FDCB..5E

COMANDO	CÓDIGO	COMANDO	CÓDIGO
BIT 3,A	CB5F	BIT 7,L	CB7D
BIT 3,B	CB58	CALL C,NN	DC
BIT 3,C	CB59	CALL M,NN	FC
BIT 3,D	CB5A	CALL NC,NN	D4
BIT 3,E	CB5B	CALL NN	CD
BIT 3,H	CB5C	CALL NZ,NN	C4
BIT 3,L	CB5D	CALL P,NN	F4
BIT 4,(HL)	CB66	CALL PE,NN	EC
BIT 4,(IX+IND)	DDCB..66	CALL PO,NN	E4
BIT 4,(IY+IND)	FDCB..66	CALL Z,NN	CC
BIT 4,A	CB67	CCF	3F
BIT 4,B	CB60	CP (HL)	BE
BIT 4,C	CB61	CP (IX+IND)	DDBE
BIT 4,D	CB62	CP (IY+IND)	FDBE
BIT 4,E	CB63	CP A	BF
BIT 4,H	CB64	CP B	B8
BIT 4,L	CB65	CP C	B9
BIT 5,(HL)	CB6E	CP D	BA
BIT 5,(IX+IND)	DDCB..6E	CP E	BB
BIT 5,(IY+IND)	FDCB..6E	CP H	BC
BIT 5,A	CB6F	CP L	BD
BIT 5,B	CB68	CP N	FE
BIT 5,C	CB69	CPD	EDA9
BIT 5,D	CB6A	CPDR	EDB9
BIT 5,E	CB6B	CPI	EDA1
BIT 5,H	CB6C	CPIR	EDB1
BIT 5,L	CB6D	CPL	2F
BIT 6,(HL)	CB76	DAA	27
BIT 6,(IX+IND)	DDCB..76	DEC (HL)	35
BIT 6,(IY+IND)	FDCB..76	DEC (IX+IND)	DD35
BIT 6,A	CB77	DEC (IY+IND)	FD35
BIT 6,B	CB70	DEC A	3D
BIT 6,C	CB71	DEC B	05
BIT 6,D	CB72	DEC BC	0B
BIT 6,E	CB73	DEC C	0D
BIT 6,H	CB74	DEC D	15
BIT 6,L	CB75	DEC DE	1B
BIT 7,(HL)	CB7E	DEC E	1D
BIT 7,(IX+IND)	DDCB..7E	DEC H	25
BIT 7,(IY+IND)	FDCB..7E	DEC HL	2B
BIT 7,A	CB7F	DEC IX	DD2B
BIT 7,B	CB78	DEC IY	FD2B
BIT 7,C	CB79	DEC L	2D
BIT 7,D	CB7A	DEC SP	3B
BIT 7,E	CB7B	DI	F3
BIT 7,H	CB7C	DJNZ DIS	10

COMANDO	CÓDIGO	COMANDO	CÓDIGO
EI	FB	JP NZ,NN	C2
EX (SP),HL	E3	JP P,NN	F2
EX (SP),IX	DDE3	JP PE,NN	EA
EX (SP),IY	FDE3	JP PO,NN	E2
EX AF,AF'	08	JP Z,NN	CA
EX DE,HL	EB	JR C,DIS	38
EXX	D9	JR DIS	18
HALT	76	JR NC,DIS	30
IM Ø	ED46	JR NZ,DIS	20
IM 1	ED56	JR Z,DIS	28
IM 2	ED5E	LD (BC),A	02
IN A,(C)	ED78	LD (DE),A	12
IN A,N	DB	LD (HL),A	77
IN B,(C)	ED40	LD (HL),B	70
IN C,(C)	ED48	LD (HL),C	71
IN D,(C)	ED50	LD (HL),D	72
IN E,(C)	ED58	LD (HL),E	73
IN H,(C)	ED60	LD (HL),H	74
IN L,(C)	ED68	LD (HL),L	75
INC (HL)	34	LD (HL),N	36
INC (IX+IND)	DD34	LD (IX+IND),A	DD77
INC (IY+IND)	FD34	LD (IX+IND),B	DD70
INC A	3C	LD (IX+IND),C	DD71
INC B	04	LD (IX+IND),D	DD72
INC BC	03	LD (IX+IND),E	DD73
INC C	0C	LD (IX+IND),H	DD74
INC D	14	LD (IX+IND),L	DD75
INC DE	13	LD (IX+IND),N	DD36
INC E	1C	LD (IY+IND),A	FD77
INC H	24	LD (IY+IND),B	FD70
INC HL	23	LD (IY+IND),C	FD71
INC IX	DD23	LD (IY+IND),D	FD72
INC IY	FD23	LD (IY+IND),E	FD73
INC L	2C	LD (IY+IND),H	FD74
INC SP	33	LD (IY+IND),L	FD75
IND	EDAA	LD (IY+IND),N	FD36
INDR	EDBA	LD (NN),A	32
INI	EDA2	LD (NN),BC	ED43
INIR	EDB2	LD (NN),DE	ED53
JP (HL)	E9	LD (NN),HL	22 ou ED63
JP (IX)	DDE9	LD (NN),IX	DD22
JP (IY)	FDE9	LD (NN),IY	FD22
JP C,NN	DA	LD (NN),SP	ED73
JP M,NN	FA	LD A,(BC)	ØA
JP NC,NN	D2	LD A,(DE)	1A
JP NN	C3	LD A,(HL)	7E

COMANDO	CÓDIGO	COMANDO	CÓDIGO
LD A,(IX+IND)	DD7E	LD D,L	55
LD A,(IY+IND)	FD7E	LD D,N	16
LD A,(NN)	3A	LD DE,(NN)	ED5B
LD A,A	7F	LD DE,NN	11
LD A,B	78	LD E,(HL)	5E
LD A,C	79	LD E,(IX+IND)	DD5E
LD A,D	7A	LD E,(IY+IND)	FD5E
LD A,E	7B	LD E,A	5F
LD A,H	7C	LD E,B	58
LD A,I	ED57	LD E,C	59
LD A,L	7D	LD E,D	5A
LD A,N	3E	LD E,E	5B
LD A,R	ED5F	LD E,H	5C
LD B,(HL)	46	LD E,L	5D
LD B,(IX+IND)	DD46	LD E,N	1E
LD B,(IY+IND)	FD46	LD H,(HL)	66
LD B,A	47	LD H,(IX+IND)	DD66
LD B,B	40	LD H,(IY+IND)	FD66
LD B,C	41	LD H,A	67
LD B,D	42	LD H,B	60
LD B,E	43	LD H,C	61
LD B,H	44	LD H,D	62
LD B,L	45	LD H,E	63
LD B,N	06	LD H,H	64
LD BC,(NN)	ED4B	LD H,L	65
LD BC,NN	01	LD H,N	26
LD C,(HL)	4E	LD HL,(NN)	2A ou ED6B
LD C,(IX+IND)	DD4E	LD HL,NN	21
LD C,(IY+IND)	FD4E	LD I,A	ED47
LD C,A	4F	LD IX,(NN)	DD2A
LD C,B	48	LD IX,NN	DD21
LD C,C	49	LD IY,(NN)	FD2A
LD C,D	4A	LD IY,NN	FD21
LD C,E	4B	LD L,(HL)	6E
LD C,H	4C	LD L,(IX+IND)	DD6E
LD C,L	4D	LD L,(IY+IND)	FD6E
LD C,N	0E	LD L,A	6F
LD D,(HL)	56	LD L,B	68
LD D,(IX+IND)	DD56	LD L,C	69
LD D,(IY+IND)	FD56	LD L,D	6A
LD D,A	57	LD L,E	6B
LD D,B	50	LD L,H	6C
LD D,C	51	LD L,L	6D
LD D,D	52	LD L,N	2E
LD D,E	53	LD R,A	ED4F
LD D,H	54	LD SP,(NN)	ED7B

COMANDO	CÓDIGO	COMANDO	CÓDIGO
LD SP,HL	F9	RES 0,(IX+IND)	DDCB..8E
LD SP,IX	DDF9	RES 0,(IY+IND)	FDCB..8E
LD SP,IY	FDF9	RES 0,A	CB87
LD SP,NN	31	RES 0,B	CB80
LDD	EDA8	RES 0,C	CB81
LDDR	EDB8	RES 0,D	CB82
LDI	EDA0	RES 0,E	CB83
LDIR	EDB0	RES 0,H	CB84
NEG	ED44	RES 0,L	CB85
NOP	00	RES 1,(HL)	CB8E
OR (HL)	B6	RES 1,(IX+IND)	DDCB..8E
OR (IX+IND)	DDB6	RES 1,(IY+IND)	FDCB..8E
OR (IY+IND)	FDB6	RES 1,A	CB8F
OR A	B7	RES 1,B	CB88
OR B	B0	RES 1,C	CB89
OR C	B1	RES 1,D	CB8A
OR D	B2	RES 1,E	CB8B
OR E	B3	RES 1,H	CB8C
OR H	B4	RES 1,L	CB8D
OR L	B5	RES 2,(HL)	CB96
OR N	F6	RES 2,(IX+IND)	DDCB..96
OTDR	EDBB	RES 2,(IY+IND)	FDCB..96
OTIR	EDB3	RES 2,A	CB97
OUT (C),A	ED79	RES 2,B	CB90
OUT (C),B	ED41	RES 2,C	CB91
OUT (C),C	ED49	RES 2,D	CB92
OUT (C),D	ED51	RES 2,E	CB93
OUT (C),E	ED59	RES 2,H	CB94
OUT (C),H	ED61	RES 2,L	CB95
OUT (C),L	ED69	RES 3,(HL)	CB9E
OUT N,A	D3	RES 3,(IX+IND)	DDCB..9E
OUTD	EDA8	RES 3,(IY+IND)	FDCB..9E
OUTI	EDA3	RES 3,A	CB9F
POP AF	F1	RES 3,B	CB98
POP BC	C1	RES 3,C	CB99
POP DE	D1	RES 3,D	CB9A
POP HL	E1	RES 3,E	CB9B
POP IX	DDE1	RES 3,H	CB9C
POP IY	FDE1	RES 3,L	CB9D
PUSH AF	F5	RES 4,(HL)	CBA6
PUSH BC	C5	RES 4,(IX+IND)	DDCB..A6
PUSH DE	D5	RES 4,(IY+IND)	FDCB..A6
PUSH HL	E5	RES 4,A	CBA7
PUSH IX	DDE5	RES 4,B	CBA0
PUSH IY	FDE5	RES 4,C	CBA1
RES 0,(HL)	CB86	RES 4,D	CBA2

COMANDO	CÓDIGO	COMANDO	CÓDIGO
RES 4,E	CBA3	RL (IY+IND)	FDCB..16
RES 4,H	CBA4	RL A	CB17
RES 4,L	CBA5	RL B	CB10
RES 5,(HL)	CBAE	RL C	CB11
RES 5,(IX+IND)	DDCB..AE	RL D	CB12
RES 5,(IY+IND)	FDCB..AE	RL E	CB13
RES 5,A	CBAF	RL H	CB14
RES 5,B	CBA8	RL L	CB15
RES 5,C	CBA9	RLA	17
RES 5,D	CBAAA	RLC (HL)	CB06
RES 5,E	CBAB	RLC (IX+IND)	DDCB..06
RES 5,H	CBAC	RLC (IY+IND)	FDCB..06
RES 5,L	CBAD	RLCA	CB07
RES 6,(HL)	CBB6	RLCB	CB00
RES 6,(IX+IND)	DDCB..B6	RLCC	CB01
RES 6,(IY+IND)	FDCB..B6	RLCD	CB02
RES 6,A	CB87	RLCE	CB03
RES 6,B	CBB0	RLCH	CB04
RES 6,C	CBB1	RLCL	CB05
RES 6,D	CBB2	RLCA	07
RES 6,E	CBB3	RLD	ED6F
RES 6,H	CBB4	RR (HL)	CB1E
RES 6,L	CBB5	RR (IX+IND)	DDCB..1E
RES 7,(HL)	CBBE	RR (IY+IND)	FDCB..1E
RES 7,(IX+IND)	DDCB..BE	RR A	CB1F
RES 7,(IY+IND)	FDCB..BE	RR B	CB18
RES 7,A	CBBF	RR C	CB19
RES 7,B	CBB8	RR D	CB1A
RES 7,C	CBB9	RR E	CB1B
RES 7,D	CBBA	RR H	CB1C
RES 7,E	CBBB	RR L	CB1D
RES 7,H	CBBC	RRA	1F
RES 7,L	CBBD	RRC (HL)	CB0E
RET	C9	RRC (IX+IND)	DDCB..0E
RET C	D8	RRC (IY+IND)	FDCB..0E
RET M	F8	RRC A	CB0F
RET NC	D0	RRC B	CB08
RET NZ	C0	RRC C	CB09
RET P	F0	RRC D	CB0A
RET PE	E8	RRC E	CB0B
RET PO	E0	RRC H	CB0C
RET Z	C8	RRC L	CB0D
RETI	ED4D	RRCA	0F
RETN	ED45	RRD	ED67
RL (HL)	CB16	RST 0	C7
RL (IX+IND)	DDCB..16	RST 8	CF

COMANDO	CÓDIGO	COMANDO	CÓDIGO
RST 10h	D7	SET 2,B	CBD0
RST 18h	DF	SET 2,C	CBD1
RST 20h	E7	SET 2,D	CBD2
RST 28h	EF	SET 2,E	CBD3
RST 30h	F7	SET 2,H	CBD4
RST 38h	FF	SET 2,L	CBD5
SBC A,(HL)	9E	SET 3,(HL)	CBDE
SBC A,(IX+IND)	DD9E	SET 3,(IX+IND)	DDCB..DE
SBC A,(IY+IND)	FD9E	SET 3,(IY+IND)	FDCB..DE
SBC A,A	9F	SET 3,A	CBDF
SBC A,B	98	SET 3,B	CBD8
SBC A,C	99	SET 3,C	CBD9
SBC A,D	9A	SET 3,D	CBDA
SBC A,E	9B	SET 3,E	CBDB
SBC A,H	9C	SET 3,H	CBDC
SBC A,L	9D	SET 3,L	CBDD
SBC A,N	DE	SET 4,(HL)	CBE6
SBC HL,BC	ED42	SET 4,(IX+IND)	DDCB..E6
SBC HL,DE	ED52	SET 4,(IY+IND)	FDCB..E6
SBC HL,HL	ED62	SET 4,A	CBE7
SBC HL,SP	ED72	SET 4,B	CBE0
SCF	37	SET 4,C	CBE1
SET 0,(HL)	CBC6	SET 4,D	CBE2
SET 0,(IX+IND)	DDCB..C6	SET 4,E	CBE3
SET 0,(IY+IND)	FDCB..C6	SET 4,H	CBE4
SET 0,A	CBC7	SET 4,L	CBE5
SET 0,B	CBC0	SET 5,(HL)	CBE6
SET 0,C	CBC1	SET 5,(IX+IND)	DDCB..EE
SET 0,D	CBC2	SET 5,(IY+IND)	FDCB..EE
SET 0,E	CBC3	SET 5,A	CBEF
SET 0,H	CBC4	SET 5,B	CBE8
SET 0,L	CBC5	SET 5,C	CBE9
SET 1,(HL)	CBCE	SET 5,D	CBEA
SET 1,(IX+IND)	DDCB..CE	SET 5,E	CBEB
SET 1,(IY+IND)	FDCB..CE	SET 5,H	CBEC
SET 1,A	CBCF	SET 5,L	CBED
SET 1,B	CBC8	SET 6,(HL)	CBF6
SET 1,C	CBC9	SET 6,(IX+IND)	DDCB..F6
SET 1,D	CBCA	SET 6,(IY+IND)	FDCB..F6
SET 1,E	CBCB	SET 6,A	CBF7
SET 1,H	CBCC	SET 6,B	CBF0
SET 1,L	CBCD	SET 6,C	CBF1
SET 2,(HL)	CBD6	SET 6,D	CBF2
SET 2,(IX+IND)	DDCB..D6	SET 6,E	CBF3
SET 2,(IY+IND)	FDCB..D6	SET 6,H	CBF4
SET 2,A	CBD7	SET 6,L	CBF5

COMANDO	CÓDIGO	COMANDO	CÓDIGO
SET 7,(HL)	CBFE	SRL (IX+IND)	DDCB..3E
SET 7,(IX+IND)	DDCB..FE	SRL (IY+IND)	FDCB..3E
SET 7,(IY+IND)	FDCB..FE	SRL A	CB3F
SET 7,A	CBFF	SRL B	CB38
SET 7,B	CBF8	SRL C	CB39
SET 7,C	CBF9	SRL D	CB3A
SET 7,D	CBFA	SRL E	CB3B
SET 7,E	CBFB	SRL H	CB3C
SET 7,H	CBFC	SRL L	CB3D
SET 7,L	CBFD	SUB (HL)	96
SLA (HL)	CB26	SUB (IX+IND)	DD96
SLA (IX+IND)	DDCB..26	SUB (IY+IND)	FD96
SLA (IY+IND)	FDCB..26	SUB A	97
SLA A	CB27	SUB B	90
SLA B	CB20	SUB C	91
SLA C	CB21	SUB D	92
SLA D	CB22	SUB E	93
SLA E	CB23	SUB H	94
SLA H	CB24	SUB L	95
SLA L	CB25	SUB N	D6
SRA (HL)	CB2E	XOR (HL)	AE
SRA (IX+IND)	DDCB..2E	XOR (IX+IND)	DDAE
SRA (IY+IND)	FDCB..2E	XOR (IY+IND)	FDAE
SRA A	CB2F	XOR A	AF
SRA B	CB28	XOR B	A8
SRA C	CB29	XOR C	A9
SRA D	CB2A	XOR D	AA
SRA E	CB2B	XOR E	AB
SRA H	CB2C	XOR H	AC
SRA L	CB2D	XOR L	AD
SRL (HL)	CB3E	XOR N	EE

Só estão listadas as instruções que afetam alguma flag. Usamos os seguintes códigos:

- * flag afetada, resultado depende da operação
- ? flag afetada, resultado indefinido
- Ø flag afetada, resultado Ø
- 1 flag afetada, resultado 1
- flag não afetada
- X ver observação
- r registrador simples
- rr par de registradores
- n número de 1 byte
- nn número de 2 bytes

INSTRUÇÃO REGISTRADOR F

	S	Z	Ø	H	Ø	P/V	N	C
ADC A,r	*	*	-	*	-	*	Ø	*
ADC HL,rr	*	*	-	*	-	*	Ø	*
ADD A,r	*	*	-	*	-	*	Ø	*
ADD HL,rr	-	-	-	*	-	-	Ø	*
ADD IX,rr	-	-	-	*	-	-	Ø	*
ADD IY,rr	-	-	-	*	-	-	Ø	*
AND r	*	*	-	1	-	*	Ø	Ø
BIT n,r	?	*	-	1	-	*	Ø	-
CCF	-	-	-	X	-	-	Ø	*
CP r	*	*	-	*	-	*	1	*
CPD	*	X	-	*	-	X	1	-
CPDR	*	X	-	*	-	X	1	-

A flag H assume o valor anterior da flag C.

Se A = (HL) então
Z = 1

Se BC = Ø então
P/V = Ø

INSTRUÇÃO REGISTRADOR F

	S	Z	0	H	0	P/V	N	C
CPI	*	X	-	*	-	X	1	-
CPIR	*	X	-	*	-	X	1	-
CPL	-	-	-	1	-	-	1	-
DAA	*	*	-	*	-	*	-	*
DEC r	*	*	-	*	-	*	1	-
IN r,(C)	*	*	-	*	-	*	0	-
INC r	*	*	-	*	-	*	0	-
IND	?	X	-	?	-	?	1	-
INDR	?	1	-	?	-	?	1	-
INI	?	X	-	?	-	?	1	-
INIR	?	1	-	?	-	?	1	-
LD A,I	*	*	-	0	-	X	0	-
LD A,R	*	*	-	0	-	X	0	-
LDD	-	-	-	0	-	X	0	-
LDDR	-	-	-	0	-	0	0	-
LDI	-	-	-	0	-	X	0	-
LDIR	-	-	-	0	-	0	0	-
NEG	*	*	-	*	-	*	1	*
OR	*	*	-	0	-	*	0	0
OTDR	?	1	-	?	-	?	1	-
OTIR	?	1	-	?	-	?	1	-
OUTD	?	X	-	?	-	?	1	-

Se BC = 0 então
P/V = 0
Se A = (HL) então
Z = 1

Se B = 0 então Z = 1

Se B = 0 então Z = 1

O conteúdo da "flag"
de interrupção é
copiado na flag P/V

Se BC = 0 então
P/V = 0

Se BC = 0 então
P/V = 0

Se B = 0 então Z = 1

INSTRUÇÃO REGISTRADOR F

	S	Z	0	H	0	P/V	N	C
OUTI	?	X	-	?	-	?	1	1
POP AF	X	X	X	X	X	X	X	X
RL r	*	*	-	0	-	*	0	*
RLA	-	-	-	0	-	-	0	*
RLC r	*	*	-	0	-	*	0	*
RLCA	-	-	-	0	-	-	0	*
RLD	*	*	-	0	-	*	0	-
RR r	*	*	-	0	-	*	0	*
RRA	-	-	-	0	-	-	0	*
RRC r	*	*	-	0	-	*	0	*
RRC A	-	-	-	0	-	-	0	*
RRD	*	*	-	0	-	*	0	-
SBC A,r	*	*	-	*	-	*	1	-
SBC HL,rr	*	*	-	*	-	*	1	*
SCF	-	-	-	0	-	-	0	1
SLA r	*	*	-	0	-	*	0	*
SRA r	*	*	-	0	-	*	0	*
SRL r	*	*	-	0	-	*	0	*
SUB r	*	*	-	*	-	*	1	*
XOR r	*	*	-	0	-	*	0	0

Se B = 0 então Z = 1

As flags são determinadas pelo byte no topo da pilha

Suponhamos que você comprou uma fita com um jogo e, observando a listagem, lá está a linha REM com códigos de máquina, e você quer desmontar as rotinas ali existentes. Para isto é necessário obter os códigos hexadecimais. Nada mais fácil.

Mantendo o programa no computador, digite este programinha:

```
9000 DIM C(2)
9005 DIM C$(2)
9010 LET E=16514
9015 LET N=PEEK E
9020 LET C(1)=INT (N/16)
9025 LET R=N-C (1) * 16
9030 LET C(2)=R
9035 FOR A=1 TO 2
9040 LET C$ (A)=CHR$ (C(A)+28)
9045 NEXT A
9050 PRINT E; "---->";C$, CHR$ N
9055 LET E=E+1
9060 GOTO 9015
```

Observações:

- 1º) Para rodar o programa, digite RUN 9000 e NEW LINE.
- 2º) Coloque este programa onde ele couber; assim, o número 9000 da linha inicial é apenas uma sugestão.
- 3º) Naturalmente é possível alterar o endereço inicial 16514 (linha 9010) de acordo com as suas necessidades.
- 4º) Quando a tela “encher”, o programa pára com denotação 5/9050; simplesmente digite CONT e NEW LINE.
- 5º) Naturalmente (fazendo E=0) é possível ver os códigos de máquina da ROM, o maior programa em linguagem de máquina.
- 6º) Consulte o Apêndice 14 para outros endereços iniciais dentro da ROM.

Antes de mais nada, o melhor carregador ASSEMBLY é o que *você escrever*, pois vai adaptar-se ao seu jeito. Programação tem muito de pessoal.

Este é o carregador ASSEMBLY que *eu uso*, pois gosto de poder entrar com *quantos códigos hexa eu quiser ao mesmo tempo*. Dá maior velocidade, principalmente em programas longos. O programa é assim:

1 REM quantos caracteres forem necessários

modo direto: POKE 16510, 0 para transformar a linha 1 em linha 0

```
10 LET E = 16514
20 LET H$ = ""
30 IF H$ = "" THEN INPUT H$
40 IF CODE H$ > CODE "F" OR CODE H$ < CODE "0" THEN STOP
50 PRINT E; "---->; H$ ( TO 2)
60 POKE E, 16*CODE H$ + CODE H$ (2) - 476
70 LET E = E + 1
80 LET H$ = H$ (3 TO )
90 IF PEEK 16442 = 2 THEN CLS
100 GOTO 30
```

Observações:

linha 10 → às vezes, é necessário alterar o endereço inicial. Vários programas do Capítulo 3 têm outro endereço inicial. Assim sendo, altere convenientemente...

linha 20 → aspas aspas, sem nenhum espaço entre elas.

linha 30 → idem.

linha 40 → para parar o programa, digite qualquer caractere que *não* possa ser o primeiro de um código hexa; por exemplo: G.

linha 50 → teclas J e M com shift.

linha 90 → no endereço 16442 está a variável do sistema que controla a linha de impressão. Se temos PRINT AT 0, ...; então PEEK 16442 = 23 (0 + 23 = 23). Assim, quando a impressão ocorrer na linha 21, PEEK 16442 = 2 (21 + 2 = 23); e a tela é limpa (CLS). Isto evita o "erro 5".

ENDEREÇO		ROTINAS
DECIMAL	HEXA	
0	0000	Inicializa o sistema (cold start); RST 0; RAND USR 0
8	0008	Manipula erros; RST 8
16	0010	Imprime o caractere cujo código está no acumulador; RST 10h
24	0018	Carrega o acumulador com o byte apontado pela variável do sistema CH.ADD; RST 18h
32	0020	Análoga à anterior, mas com o próximo byte; RST 20h
40	0028	Organiza cálculos com ponto flutuante; RST 28h
48	0030	Incrementa a área das variáveis com o número de bytes determinado por BC; RST 30h
56	0038	Gera o display, é operada por hardware; RST 38h
102	0066	Gera o display em modo SLOW
126	007E	Tabela dos caracteres normais do teclado
165	00A5	Tabela dos caracteres SHIFT do teclado
204	00CC	Tabela das funções do BASIC
243	00F3	Tabela dos caracteres gráficos
273	0111	Tabela das palavras chaves (KEY-WORDS) do BASIC
508	01FC	Atualiza os comandos SAVE e LOAD
519	0207	Determina SLOW ou FAST
553	0229	Principal rotina do display
658	0292	Display em modo SLOW
693	02B5	Display em modo FAST
699	02BB	Varre o teclado (KEYBOARD SCAN)
756	02F4	Comando SAVE; ver Apêndice 16
832	0340	Comando LOAD; ver Apêndice 16
930	03A2	Teste de BREAK no comando LOAD
963	03C3	Comando NEW = RAND USR 963
1049	0419	Edição das linhas de programação
1108	0454	Rotina do cursor
1154	0482	Constrói o sistema da variável E.LINE
1323	052B	Ordena a edição
1476	05C4	Principal rotina de edição
1598	063E	Execução do programa BASIC
1836	072C	Comando LLIST
1840	0730	Comando LIST = RAND USR 1840
1861	0745	Impressão de uma linha BASIC
1981	07BD	Decodificação do teclado; FIND CHARACTER; normalmente usada em conjunção com KEYBOARD SCAN

ENDEREÇO		ROTINAS
DECIMAL	HEXA	
2033	07F1	Impressão de um caractere
2056	0808	Impressão de um caractere
2129	0851	Carga de um caractere no buffer da impressora
2153	0869	Comando COPY
2293	08F5	Testa os parâmetros do PRINT AT
2328	0918	Expansão do display
2379	094B	Impressão das palavras-chave do BASIC
2462	099E	Expansão do programa BASIC (conseqüentemente D.FILE e VARS)
2477	09AD	Organização das variáveis
2520	09D8	Determinação do endereço de uma linha de programa
2602	0A2A	Comando CLS = RAND USR 2602
2712	0A98	Impressão do número da linha do BASIC
2763	0ACB	Comando LPRINT
2767	0ACF	Comando PRINT
2923	0B6B	Impressão de uma string
2991	0BAF	Comandos PLOT e UNPLOT
3086	0C0E	Comando SCROLL = RAND USR 3086
3113	0C29	Tabela de sintaxe dos comandos
3292	0CDC	Comando STOP
3434	0D6A	Comando REM
3499	0DAB	Comando IF
3513	0DB9	Comando FOR
3630	0E2E	Comando NEXT
3692	0E6C	Comando RAND
3708	07EC	Comando CONT
3713	0E81	Comando GOTO
3730	0E92	Comando POKE
3759	0EAF	Comando RUN = RAND USR 3759
3765	0EB5	Comando GOSUB
3800	0ED8	Comando RETURN
3817	0EE9	Comando INPUT
3875	0F23	Comando FAST = RAND USR 3875
3883	0F2B	Comando SLOW = RAND USR 3883
3890	0F32	Comando PAUSE
3910	0F46	Teste de BREAK do comando SAVE
4897	1321	Comando LET
5129	1409	Comando DIM
5274	149A	Comando CLEAR
5405	151D	Arquiva acumulador na pilha do calculador
5408	1520	Arquiva o par de registradores BC na pilha do calculador
5514	158A	Manipulação dos cálculos de ponto flutuante
5964	174C	Subtração de números de 5 bytes

ENDEREÇO		ROTINAS
DECIMAL	HEXA	
5973	1755	Adição de números de 5 bytes
6086	17C6	Multiplicação de números de 5 bytes
6274	1882	Divisão de números de 5 bytes
6421	1915	Tabela de funções
6557	199D	Calculador de ponto flutuante
7680	1E00	Tabela de definição dos caracteres

Verifique se há algum byte errado na ROM de seu micro, rodando o seguinte programa. A resposta leva pouco mais de um minuto para surgir na tela (apesar do programa estar em FAST).

```
10 FAST
20 LET S=0
30 FOR E=0 TO 8191
40 LET S=S+PEEK E
50 NEXT E
60 SLOW
70 PRINT "SOMA = ";S
```

A tabela abaixo fornece os resultados da soma dos conteúdos de todos os endereços da ROM para cada micro de lógica SINCLAIR.

EQUIPAMENTO	SOMA
NE-Z8000	855106
CP-200	855106 ou 855660
TK 82-C	855106 ou 854169
TK 85	854169

Observação: Nos casos de dois resultados, o primeiro deles se refere a equipamentos mais antigos; o segundo aos modelos mais recentes.

Aqui colocamos várias observações que não "cabiam" nos apêndices anteriores.

- 1^a) RST Ø é ativada quando o micro é ligado; inicializa todo o sistema; logo a digitação de RAND USR Ø equivale a desligar e religar o computador. Se você tem acesso ao teclado (não houve CRASH), é preferível digitar RAND USR Ø do que desligar e ligar. O efeito de RST Ø é mais "forte" do que NEW, pois RAMTOP é colocado no seu valor máximo.
- 2^a) Alguns programas escritos totalmente em linguagem de máquina não são "sensíveis" à tecla BREAK. Quando carregados a partir da fita, saem "rodando" automaticamente e "não é possível" pará-los ou listá-los.

No entanto, isto é possível. Sabemos que o comando LOAD começa identificando o nome do programa e, a partir disto, inicia a gravação da RAM.

Assim sendo, o que temos de fazer é "saltar" a verificação do nome, introduzindo um erro no sistema. Logo, ao final da gravação, o micro acusa erro (C/Ø), e você pode confortavelmente listar o programa. Como fazer isto?

O comando LOAD começa no endereço 832 (ver Apêndice 14), e podemos tentar entrar na rotina até o endereço 842, o que melhor funciona é:

- a) DIGITE FAST (NEW LINE)
 - b) DIGITE RAND USR 837 (NEW LINE)
 - c) APÓS A CARGA, O PROGRAMA PÁRA COM C/Ø
 - d) LISTE O PROGRAMA...
- 3^a) No entanto, você pode defender seus programas! Se um dia você escrever um programa comercial totalmente em linguagem de máquina, e, portanto, "insensível" à tecla BREAK, defenda-o do "RAND USR 837" que ensinamos acima!

RAND USR 837 pára o programa por não ter verificado o nome, introduzindo um erro no programa.

Assim, precisamos *gravar* o programa SEM NOME! Como isto é possível? Assim:

— você gravava da seguinte forma:

999Ø	SAVE "NOME"
9995	RUN
GOTO 999Ø	seguido de NEW LINE

— passe a gravar assim:

998Ø	FAST
9985	RAND USR 757
999Ø	SLOW
9995	RUN
GOTO 998Ø	seguido de NEW LINE

Para carregar o programa, gravado desta maneira, LOAD "" e NEW LINE. O programa não mais será "parado" por RAND USR 837...

Impressão e Acabamento
GRAPHOS
Rua Santo Cristo, 70/78 — Tel.: 223-1272



Além do BASIC

Linguagem ASSEMBLY para a Linha SINCLAIR

Este é o primeiro livro "didático" sobre linguagem de máquina para micros da linha Sinclair. Evitando as "fórmulas mágicas", esclarece as dúvidas e desvenda todos os mistérios desta difícil mas fascinante linguagem.

Através de uma abordagem simples e coloquial ensina as primeiras instruções Assembly em programas de complexidade crescente, que o leitor aprende a elaborar através de grande número de exercícios cuidadosamente selecionados. As respostas dos exercícios também facilitarão o trabalho de aprender uma nova linguagem, bem como os apêndices, com material inédito no Brasil referente ao Z-80 em geral e aos micros Sinclair em particular. Um grande número de rotinas em linguagem de máquina, de estimulante aspecto visual, enriquecerão e simplificarão os programas BASIC do leitor.

Um apaixonado da computação, Nelson do Nascimento Silva dos Santos, há longos anos Professor de Química e Matemática, é responsável pela coluna *Além do Basic* na revista MICROBITS (Editora Campus) e, ainda, autor de artigos em outras revistas da área.